VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Dálkové ovládání robotů pomocí mobilního telefonu

Android Remote Controller for Small Robots and Drones

2016 Bc. David Kuna

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. David Kuna**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Dálkové ovládání robotů pomocí mobilního telefonu
Android Remote Controller for Small Robots and Drones

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit mobilní aplikaci pro dálkové řízení robotů a dronů. Práce se bude skládat z klientské ovládací aplikace (běžící na mobilním telefonu nebo tabletu s OS Android) a serveru, který může běžet taktéž na mobilním telefonu nebo jiné platformě podporující OS Android.
Serverová aplikace zabezpečuje sběr dat z vnitřních a vnějších senzorů, přenos obrazových dat ze zabudované kamery a přeposílání řídících příkazů elektronice robota.

1. Porovnání dostupných řídících systémů pro roboty a malé drony.
2. Přenos dat mezi klientem a serverem (varianty GSM/UMTS/WiFi).
3. Implementace serverové části, komunikace a sběr dat ze senzorů (výběr vhodného kódování pro video data).
4. Implementace klientské části, vizualizace získaných telemetrických dat, dekódování obrazových dat, dálkové ovládání s možností záznamu pro pozdější přehrání.
5. Otestování navrženého řešení na dostupné platformě.

Seznam doporučené odborné literatury:

[1] Reto Meier, Professional Android 4 Application Development, Wrox, 2012, ISBN-13: 978-1118102275
[2] Sayed Hashimi, Pro Android 2, Apress, 2010, ISBN-13: 978-1430226598
[3] Howie Choset et al., Principles of Robot Motion: Theory, Algorithms, and Implementations, A Bradford Book, 2005, ISBN-13: 978-0262033275

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 25. dubna 2016 .....

Rád bych na tomto místě poděkoval všem, kteří mi pomohli při tvorbě diplomové práce, především pak panu Mgr. Ing. Michalu Krumniklovi, Ph.D. za vedení této práce a ochotnou pomoc při řešení problémů.

Abstrakt

Tato diplomová práce se zabývá vývojem řídícího systému pro dálkové ovládání robotů a malých dronů pomocí mobilního telefonu s operačním systémem Android. Obsahuje uvedení do problematiky sběru dat z integrovaných senzorů v mobilním zařízení, kódování přenášeného obrazu a síťové komunikace. Dále poskytuje náhled na již dostupná řešení řídících systémů a jejich srovnání. V rámci práce vznikly dvě mobilní aplikace pro operační systém Android, které tvoří serverovou a klientskou část řídícího systému umožňujícího přenos telemetrických dat, obrazu z integrované videokamery a řídících signálů. Popis jednotlivých částí sytému je součástí textu.

Klíčová slova: Android, senzor, dálkové ovládání, diplomová práce, přenos obrazu

Abstract

This thesis is aimed at developing the remote control system for robots and small drones by smartphone with Android operating system. The first part of the thesis includes the introduction to the topic, the collecting integrated sensor data from smart devices, digital video encoding for transmission and network communication. It also provides an insight into already existing remote control systems with their comparison. In this thesis were developed two Android applications representing the client and server part of remote control system provides sensor data transmission and video streaming. Parts of each application are described in text.

Key Words: Android, sensor, remote control, master thesis, video streaming

Obsah

Seznam použitých zkratek a symbolů

JIT	–	Just-in-time
AOT	–	Ahead-of-time
OS	–	Operační systém
PC	–	Personal computer - Osobní počítač
GPS	–	Global Positioning System - Globální polohovací systém
IGMP	–	Internet Group Management Protocol
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
GMS	–	Global System for Mobile Communications
UTMS	–	Universal Mobile Telecommunications System
NAT	–	Network Address Translation
STUN	–	Session Traversal Utilities for NAT
VOIP	–	Voice over Internet Protocol
HTTP	–	Hypertext Transfer Protocol
PHP	–	PHP: Hypertext Preprocessor

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V současné době se mobilní aplikace stávají nedílnou součástí každodenního života běžných lidí. Děje se to především díky rychle rostoucímu trhu se stále výkonnějšími a cenově dostupnějšími mobilními zařízeními, vybavenými pokročilým mobilním operačním systémem. Takovými operačními systémy jsou například: iOS, Android, Windows Phone, Firefox OS anebo BlackBerry OS. V souvislosti s rozmachem mobilních zařízení je čím dál důležitější i vývoj nových aplikací, plně využívajících potenciál těchto zařízení a jejich platforem. S rostoucím výkonem zařízení, většími displeji a vyšším rozlišením je možné využívat mobilní aplikace k usnadnění činností, které by se dříve neobešly bez použití osobního počítače.

Chytré telefony tzv. „smartphony“ dnes mají v základní výbavě celou řadu senzorů. Využívají se při hraní her, k navigaci či pro automatické nastavení jasu obrazovky. I u těch nejlevnějších smartphonů nalezneme alespoň základní sadu běžných senzorů a tak se díky jejich ceně tato zařízení stávají zajímavá pro využití i k jiným účelům než jen jako mobilní telefon. Jedním z odvětví, ve kterém se dá velmi dobře využít širokých možností „low-endových“ telefonů je robotika. Mobilní telefon dokáže nahradit hned několik hlavních periferií a také se stát samotnou řídící jednotkou robota. Připojením jednoho kompaktního přístroje získá robot možnosti záznamu videa, určení jeho polohy, pohybu nebo například bezdrátové komunikace. Tímto způsobem jde velmi snadno snížit náklady na konstrukci robota. Další výhodou použití mobilního telefonu jako řídící jednotky je možnost instalace již hotového řídícího systému pro interpretaci senzorových dat a ovládání robota.

Cílem této práce je porovnat dostupné řídící systémy a vytvořit mobilní aplikaci umožňující bezdrátový přenos dat mezi mobilním telefonem umístěným na řízeném robotovi a kontrolní stanicí. Díky malým rozměrům a hmotnosti je možné připevnit mobilní telefon na malého drona, kteří jsou na trhu čím dál populárnější. Pro implementaci byla zvolena mobilní platforma Android, která má největší zastoupení mezi používanými operačními systémy pro mobilní zařízení je nasazován i do těch nejlevnějších mobilních telefonů.

2 Porovnání dostupných řídících systémů

Řídící systém je soubor jedné či více aplikací a definovaných procesů, učený k ovládání určitého druhu zařízení. V případě jedné aplikace se jedná o autonomní řídící systém, který je schopen podle předem nastavených pravidel řídit pohyb robota. Systémy pro ruční řízení se skládají z aplikace běžící na straně ovládaného zařízení „serverová aplikace“ a druhé aplikace tzv. „klientská aplikace“ běžící v kontrolní stanici. Součástí těchto systémů bývá kromě samotného ovládání také možnost získávání informací z periferií ovládaného zařízení a jejich vyhodnocování či zaznamenávání. Řídící systémy můžeme rozdělit do několika skupin podle podle různých kritérií. V této kapitole se zaměřím pouze na popis a srovnání řídících systémů pro dálkové ovládání robotů a malých dronů dostupných pro platformu Android.

2.1 Komerční systémy

Kategorie řídících systémů vyvíjených přímo výrobci robotů a dronů pro ovládání jejich vlastních sériově vyráběných zařízení. Klientská aplikace komunikuje přímo s řídícím systémem robotického zařízení a umožňuje plnou kontrolu nad jeho řízením. Tyto systémy není možné použít pro jiný druh zařízení a jejich zdrojové kódy nejsou volně šiřitelné. Výhodou je uživatelská podpora a pravidelné aktualizace přinášející nová vylepšení a opravu chyb. Z pravidla tyto drony disponují zabudovanou kamerou jejichž obraz je přenášen přes Wi-Fi nebo Bluetooth do klientského zařízení.

2.1.1 AR.Drone

Aplikace dříve známá pod názvem AR.FreeFlight je pilotní systém vyvíjený francouzskou společností Parrot SA pro dálkově řízené quadroptéry Parrot AR, které tato společnost vyrábí. Řídící systém se tedy skládá ze serverové části zabudované v těle drona a klientské aplikace AR.Drone nainstalované na mobilním zařízení. AR.Drone poskytuje pilotní funkce a umožňuje snímat fotografie a zachytávat video. Aplikace byla vydána v roce 2010 souběžně se začátkem prodeje prvního modelu drona AR.Drone a určena byla pouze pro platformu iOS.[9] Kromě režimu volného létání aplikace nabízí tutoriál pro seznámení se s funkcemi a ovládáním drona, dále obsahuje několik herních módů pro zpestření létání volným prostorem. Videa a fotografie jsou ukládány do paměti mobilního zařízení a je možné je prohlížet a spravovat přímo v aplikaci nebo nastavit automatické nahrávání do služeb YouTube a Picasa. Ve verzi 2.4 pro iOS funguje „Director Mode“, ve kterém se dají nastavit různé parametry videokamery tak aby pořízené videozáznamy dosahovaly vyšší kvality pro další zpracování. Drony AR.Drone disponují rozhraním USB a Wi-Fi a proto i řídící systém nabízí pouze možnost připojení přes bezdrátovou síť Wi-Fi. Mobilní síť může být v aplikaci využita pouze pro synchronizaci dat s cloudovými službami a stahování aktualizací.[11]

Obrázek 1: Ukázka uživatelského rozhraní aplikace AR.Drone (převzato z [10])

2.1.2 Vision Pilot

Aplikace je určena pouze pro modely dronu DJI Phantom 2 a nabízí pouze možnosti zpracování a interpretace senzorových dat a přehrávání obrazu z videokamery. Neumožňuje pilotování dronu ani jeho konfiguraci.

2.2 Systémy pro Bluetooth roboty

Je k dostání řada robotických souprav tzv. „kitů“ se zabudovaným Bluetooth rozhraním ale bez dodávaného ovládacího softwaru. Příkladem takového kitu může být DIY Remote Control Robot Kit[1] nebo Boe-Bot robotic kit[2]. Vzniklo již několik univerzálních systémů, které s těmito zařízeními dokáží komunikovat. Univerzálnost těchto systémů spočívá v možnosti konfigurace jednotlivých ovládacích prvků GUI. Systém poté robotovi zasílá příkazy ve tvaru předem definovaných hexadecimálních řetězců, kterým robot rozumí.

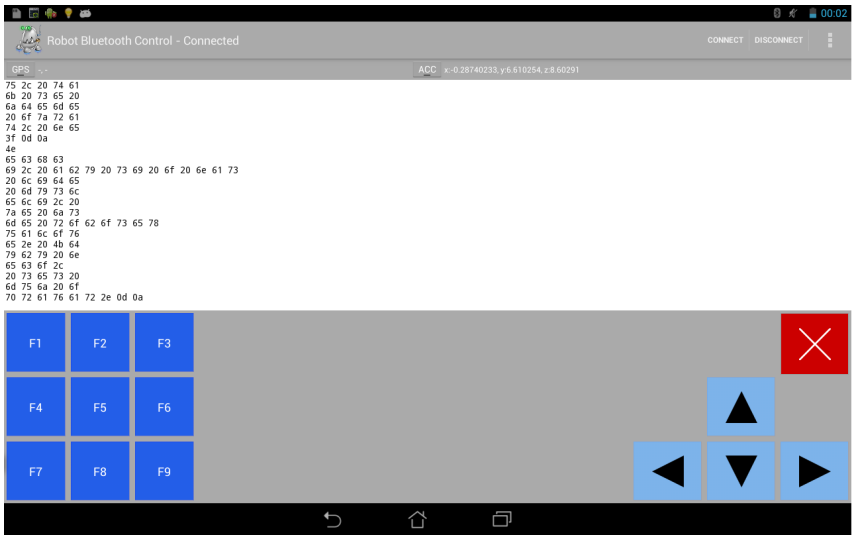
2.2.1 Robot Bluetooth Control

Tato aplikace pro platformu Android je navržena pro snadné ovládání robotů schopných přijímat příkazy přes Bluetooth Serial Port Profile. Nabízí vlastní konfiguraci tlačítek což znamená, že pro každé tlačítko je možné nastavit vlastní příkaz pro ovládané zařízení. Příkazy se zadávají jako

[1]DIY Remote Control Robot Kit (Support Android) [online] `http://www.dfrobot.com/wiki/index.php?title=DIY_Remote_Control_Robot_Kit_(Support_Android)_SKU:COMB0004`

[2]Boe-Bot Robot [online] `https://www.parallax.com/product/boe-bot-robot`

sekvence dvou znaků reprezentující hexadecimální hodnotu bajů. Stisknutím tlačítka je příkaz zaslán do předem spárovaného zařízení. Je možné odesílat také senzorická data z akcelerometru a souřadnice GPS. Aplikace je učena pro Android ve verzi 4.0 a vyšší.[13]



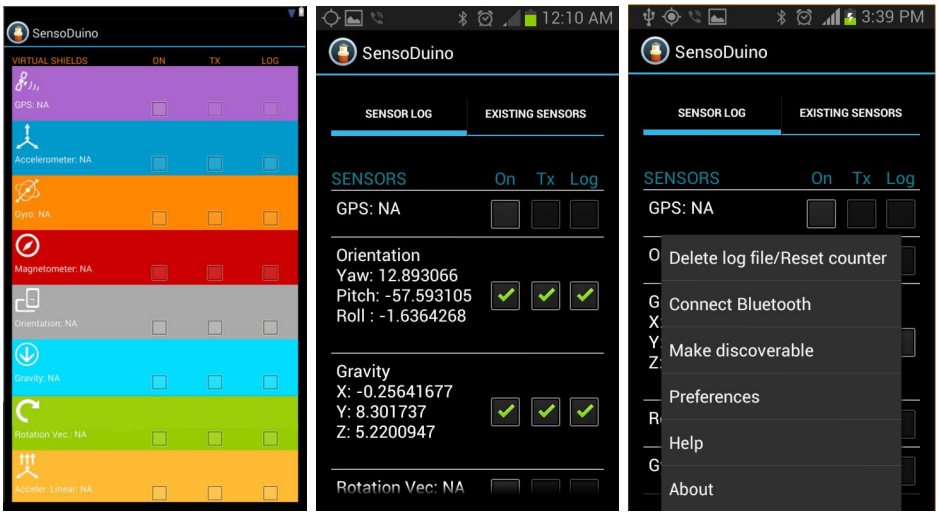
Obrázek 2: Ukázka uživatelského rozhraní aplikace Robot Bluetooth Control (převzato z [13])

2.3 Systémy pro platformu Arduino

Arduino je malý jednodeskový počítač založený na mikrokontrolerech ATmega od firmy Atmel. Je navrženo tak aby bylo jednoduše použitelné k vytváření samostatných interaktivních zařízení jako jsou roboti. Arduino je možné osadit různými moduly pro bezdrátovou komunikaci jako je Wi-Fi nebo Bluetooth. Díky tomu mohou tvořit serverovou část systému pro dálkové ovládání pomocí mobilního telefonu.

2.3.1 SensoDuino

SensoDuino zachycuje údaje ze senzorů zabudovaných v mobilním zařízení a umožňuje tato data přenášet pomocí Bluetooth. Využití se nabízí v kombinaci s platformou Arduino osazenou Bluetooth modulem HC-05. Připojením mobilního zařízení s aplikací SensoDuino je možné rozšířit možnosti robota řízeného Arduinem. Data z aplikace jde však přenášet i PC nebo jiných Bluetooth mikroprocesorů. SensoDuino také zaznamenává sesbíraná data do CSV souboru. V nastavení se dá určit frekvenci čtení dat ze senzoru, jejich odesílání a zaznamenávání. Podporována je většina integrovaných senzorů jako například GPS, akcelerometr, gyroskop, magnetometr, barometr nebo teploměr. Nevýhodou je chybějící podpora přenosu dat pomocí jiných bezdrátových technologií. Aplikace také nepodporuje záznam ani přenos obrazových dat.[14]



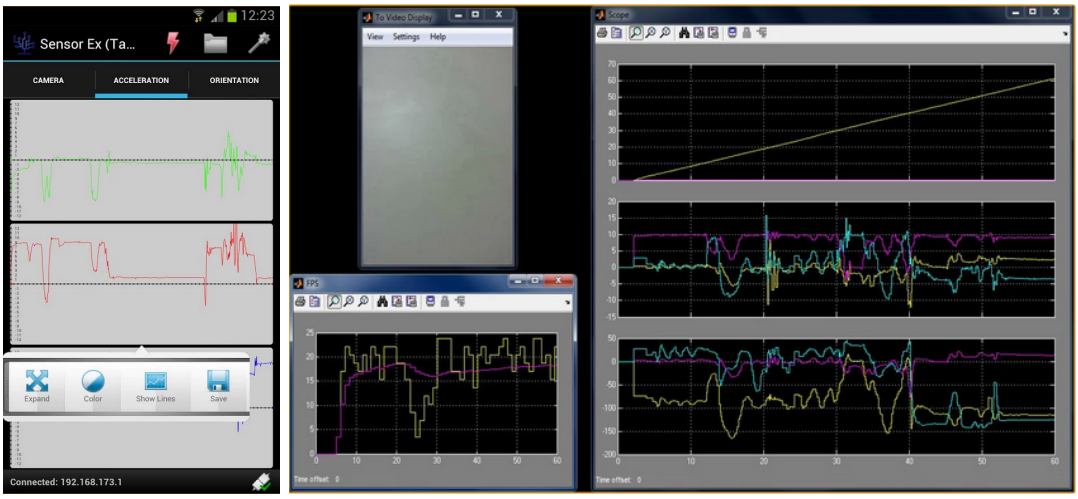
Obrázek 3: Ukázka uživatelského rozhraní aplikace SensoDuino[14]

2.4 Systémy pro streamování senzorových dat

Jedním z účelů těchto systémů je zprostředkovávat data z hardwarových i softwarových senzorů mobilního zařízení pro další aplikace, které pro svůj účel potřebují přijímat senzorová data ze vzdáleného zařízení. Nemají užší zaměření a snímaná data většinou nijak neupravují. Slouží pouze o odesílání dat a neumožňují žádné přijímání požadavků, kvůli čemuž je není možné řadit přímo mezi řídící systémy.

2.4.1 Sensor Ex

Aplikace je základní součástí systému Sensor Ex zprostředkovávajícího streamování dat v reálném čase přes Wi-Fi do PC. Systém je složen ze severové aplikace pro mobilní zařízení a klientského programu pro PC. Systém umí přenášet data z integrovaných senzorů a obrazová data z videokamery. V klientské aplikaci je možné data interpretovat pomocí grafů a tabulek s aktuálními hodnotami ze senzorů. Přijatá data je možné ukládat do CSV nebo TDAT formátu. Zaznamenávat do souboru lze také video a zvuk z videokamery. Komunikace se serverovou aplikací umožňuje pouze zapínat přídavně světlo videokamery. Součástí projektu je také .NET knihovna s demo projektem pro Visual Studio, díky které se dá celý systém přizpůsobit potřebám vlastního řešení pro zpracování přijatých dat. [16]



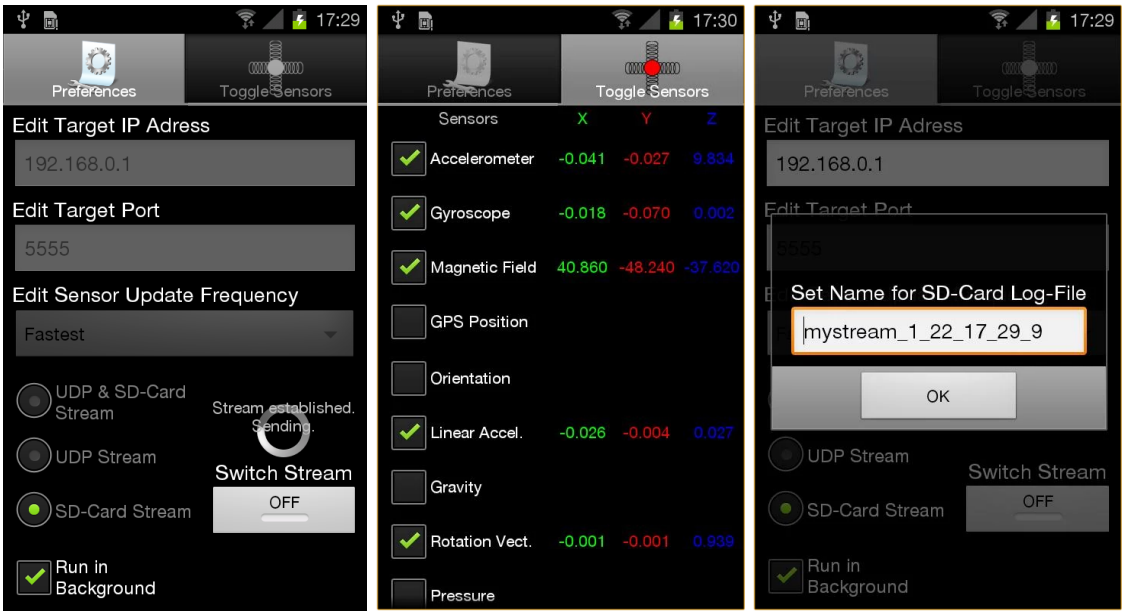
Obrázek 4: Ukázka uživatelského rozhraní systému Sensor Ex. Vlevo mobilní aplikace Sensor Ex. Vpravo klientský program pro OS Windows (převzato z [16])

2.5 DroneKit-Android

DroneKit-Android je implementací SDK DroneKit pro OS Android. Jde o soubor nástrojů a API poskytující rozhraní pro ovládání dronů, letadel a pozemních vozidel. Je kompatibilní se všemi zařízeními využívajícími protokol MAVLink. Mezi tato zařízení patří většina produktů vyráběných členy projektu DroneCode foundation. [18]

2.5.1 Sensorstream IMU+GPS

Aplikace slouží ke snímání dat z integrovaných senzorů mobilního zařízení a jejich streamování přes Wi-Fi či mobilní síť. Zvolit se dá také režim pro streamování do CSV souboru nebo obě možnosti zároveň. Aplikace snímá data ze základních senzorů a modulu GPS, který obaluje sadou algoritmů pro převod do jiných souřadnicových systémů. Stream přes WLAN využívá protokol UDP a je třeba nastavit cílovou IP adresu a port klientského zařízení. Aplikace neumožňuje přenos obrazových dat ani streamování dat pro více klientů zároveň. Výhodou je schopnost pracovat na pozadí a podpora verzí OS Android 2.3.3 a vyšší.[17]



Obrázek 5: Ukázka uživatelského rozhraní aplikace Sensorstream IMU+GPS (převzato z [17])

2.5.2 Tower

Tower je rozsáhlý řídící systém pro drony a letadla od společnosti 3D Robotics Inc., dále jen 3DR. Pro plnou funkčnost je třeba mít v mobilním zařízení nainstalovanou aplikaci 3DR Services což je interface pro DroneKit. 3DR Services tvoří společné jádro všech aplikací pro komunikaci s produkty 3DR. Aplikace Tower nabízí možnosti pro řízení stoje, sledování jeho polohy a znázornění údajů ze senzorů. Nabízí široké možnosti nastavení a funkce pro plánování a záznam trasy letu. Plánování trasy znamená, že si lze před samotným letem na mapě zaznačit body, přes které povede dráha letu tzv. „waypointy“. Speciálními značkami lze také určit místa, která má snímat videokamera během letu. Uložené záznamy letů lze zpětně analyzovat pomocí vestavěných nástrojů.

2.5.3 AndroPilot

Řídící systém pro UAV zařízení využívající platformu Ardupilot. Tato platforma je založená na jednodeskovém počítači Arduino a je v podstatě jeho rozšířením určeným pro kvadroptéry a UAV letadla. Pro jeho použití je třeba mít k mobilnímu zařízení připojený USB modul `3D Robotics telemetry`, schopný přijímat signál z ovládaného zařízení. V Evropě je pro tento signál používaná frekvence 433 MHz. Pro přenos řídících signálu a telemetrických dat se používá protokol MAVLink. Hlavními funkcemi aplikace je manuální řízení robota, monitorování jeho trasy nebo nastavení parametrů ovládaného zařízení.

2.6 MAVLink

Micro Air Vehicle Link (MAVLink) je jednoduchý protokol pro komunikaci mezi malými bezpilotními vozidly tzv. „unmanned vehicle“ a pozemní kontrolní stanicí. Protokol je bezstavový a skládá se z definovaných zpráv, výpočtových typů a ze sady předepsaných a obslužných funkcí. Protokol definuje sadu zpráv pro zjišťování stavu vozidla, zasílání ovládacích příkazů, přenos telemetrických dat a nastavování parametrů. Přenos zpráv probíhá po sériové lince, buď pomocí bezdrátového modulu 3DR Radio nebo přes USB kabel.

Vytvořená zpráva je v kontrolní stanici zakódována a odeslána v packetu do řízeného robota kde se dekóduje a spustí obslužnou funkci, které vykoná příslušné akce. Po přijetí určité zprávy může následovat odpověď odesilateli. Komunikace probíhá i opačným směrem například při zasílání telemetrických dat z robotického zařízení do kontrolní stanice. Každý packet se skládá z hlavičky (6 bajtů), těla a kontrolního součtu (2 bajty), přičemž maximální délka packetu může být 263 bajtů. Struktura packetu verze 1.0 je uvedena v tabulce 1.

Tělo packetu neboli „payload“ tvoří MAVLink zprávu. Každá zpráva je identifikovaná polem `Message ID` v hlavičce packetu. Tento identifikátor určuje definici struktury dat ve zprávě. Definice jednotlivých zpráv jsou uloženy v XML dokumentu, který je součástí knihovny MAVLink.1

Název pole	**Index**	**Význam**
Start-of-frame	0	Indikátor začátku packetu
Pay-load-length	1	Délka těla packetu (n)
Packet sequence	2	Počítadlo odeslaných sekvencí. Umožňuje detekovat ztrátu packetu
System ID	3	Identifikační číslo zařízení, které odeslalo tento packet. Slouží pro rozlišení více zařízení v dosahu.
Component ID	4	Identifikační číslo komponenty, která odeslala tento packet. Slouží k rozlišení různých komponent v jednom systému.
Message ID	5	Identifikační číslo zprávy definující typ těla packetu v poli Payload a jakým způsobem mají být data dekódována.
Payload	6 až (n+6)	Tělo zprávy jejíž obsah závisí na Message ID.
CRC	(n+7) až (n+8)	Kontrolní součet tzv. „check-sum“ určený pro detekci chyb.

Tabulka 1: Struktura packetu protokolu MAVLink

```
<message id="141" name="ALTITUDE">
   <description>The current system altitude.</description>
   <field type="uint64_t" name="time_usec">Timestamp (milliseconds since system
       boot)</field>
   <field type="float" name="altitude_monotonic">This altitude measure is
      initialized on system boot and monotonic. </field>
   <field type="float" name="altitude_amsl">This altitude measure is strictly
      above mean sea level and might be non-monotonic.</field>
   <field type="float" name="altitude_local">This is the local altitude in the
      local coordinate frame. It is not the altitude above home, but in
      reference to the coordinate origin (0, 0, 0). It is up-positive.</field>
   <field type="float" name="altitude_relative">This is the altitude above the
      home position. It resets on each change of the current home position.</
      field>
   <field type="float" name="altitude_terrain">This is the altitude above
      terrain. Values smaller than -1000 should be interpreted as unknown.</
      field>
   <field type="float" name="bottom_clearance">This is not the altitude, but
      the clear space below the system according to the fused clearance
      estimate. It generally should max out at the maximum range of e.g. the
      laser altimeter.</field>
</message>
```

Výpis 1: Definice datové zprávy protokolu MAVLink pro přenos nadmořské výšky

3 Operační systém Android

Android je operační systém pro mobilní zařízení postavený na Linuxovém jádře. Zdrojový kód je dostupný jako otevřený software tzv. „open source“ pod licencí Apache License. Android je vyvíjen konsorciem Open Handset Alliance, které má za cíl snížit náklady na vývoj a distribuci operačního systému pro mobilní zařízení. [1, 2, 7]

3.1 Historie

Historie Androidu začala v roce 2003 byla v Kalifornii založena firma Android Inc., kterou v srpnu roku 2005 odkoupil Google Inc. a udělal zní svou dceřinou společnost. Tým pod vedením jednoho ze zakladatelů Android Inc. Andym Rubinem následně vyvinul platformu založenou na Linuxovém jádře.

Koncem roku 2007 bylo sestaveno konsorcium s názvem Open Handset Alliance. Toto uskupení bylo složeno z výrobců mobilních zařízení, čipů, společností vyvíjejících mobilní aplikace, mobilních operátorů a dalších. Patří mezi ně například HTC, Samsung, Google nebo eBay. Na konci roku 2015 bylo součástí aliance 84 členů. 5. listopadu 2007 byl Android představen jako otevřená platforma postavená na jádře Linux verze 2.6. První komerční telefon s operačním systémem Android byl uveden v říjnu roku 2008 ve Spojených státech amerických. Telefon tchaj-wanské firmy HTC v dostal po uvedení název T-Mobile G1 a byl v něm použit Android verze 1.6.[1, 2]

3.2 Architektura

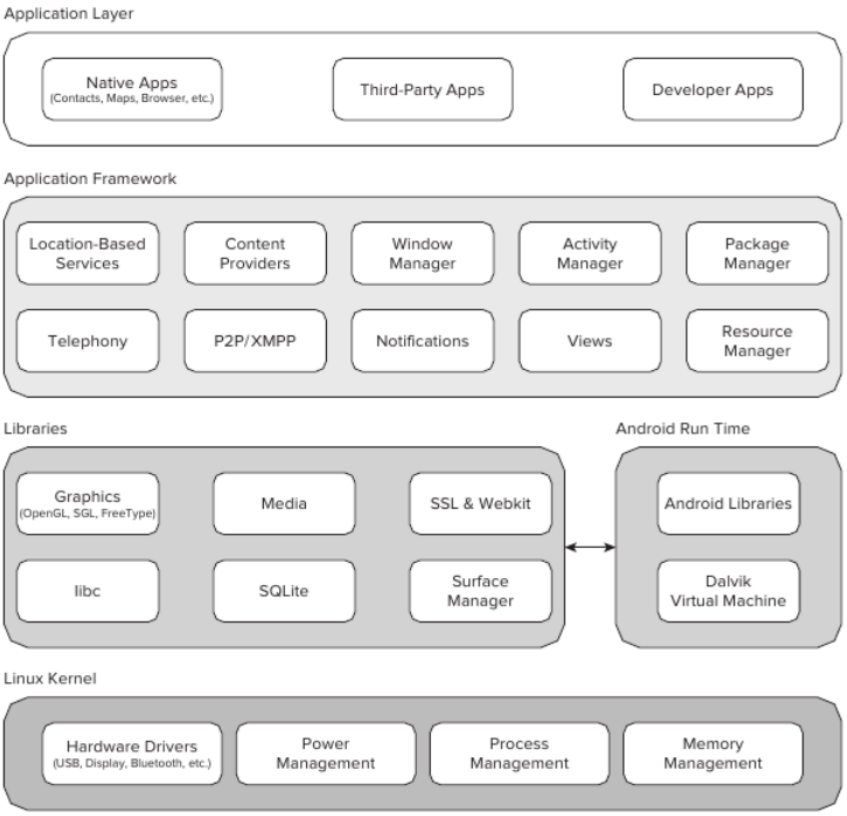
Architektura operačního systému Android se skládá z pěti vrstev, z nichž každá vrstva má pevně daný účel. Grafické znázornění všech vrstev architektury a jejich spořádání je na obrázku 6.

Nejnižší vrstvou operačního systému je systémové jádro postavené na jádře systému Linux. Tvoří abstraktní vrstvu mezi používaným hardwarem a softwarem ve vyšších vrstvách. Pracuje s hardwarovými ovladači, správou paměti, řídí souběh procesů apod. Nad touto vrstvou pracují systémové knihovny, které podporují běh různých komponent systému. Na příklad pro přehrávání videa a zvuku slouží knihovny Media Libraries, OpenGL zajišťuje renderování 3D grafiky a SQLite obaluje práci se stejnojmennou relační databází. Knihovny jsou psány v jazyce C a C++ a vývojářům poskytnuty prostřednictvím aplikačního frameworku.[1]

Dalšími knihovnami jsou knihovny jádra, které jsou společně v interpretrem obaleny vrstvou zvanou běhové prostředí. V tomto prostředí jsou spouštěny aplikace psané v programovacím jazyce Java a jsou zde také zohledňovány omezené zdroje mobilních zařízení jako je kapacita baterie, operační paměť nebo výpočetní výkon.

Do verze 4.3 Android využíval jako interpretr virtuální stroj zvaný Dalvik, který ve verzi 4.4 nahradil ART (Android Runtime).

- Dalvik Virtual Machine je virtuální stroj vyvinutý firmou Google Inc. určený podobně jako Java Virtual Machine vykonávání zkompilovaného bajtkódu. Aplikace pro Android je nejprve zkompilována do bajtkódu pro Java Virtual Machine a ten poté přeložen do Dalvik bajtkodu. Dalvik byl oproti JVM navržen tak aby mohlo v jednu chvíli efektivně běžet více instancí virtuálního stroje, přičemž každá aplikace běží na vlastní instanci virtuálního stroje.[7]
- Android Runtime je běhové prostředí nahrazující neefektivní Dalvik, fungující na principu JIT kompilace. ART používá dopřednou kompilaci (AOT - Ahead-of-time compilation) což se projevuje výrazným zrychlením aplikací a úsporou energie jelikož není třeba aplikaci překládat při každém spuštění. Kompilace a finální optimalizace probíhá jen jednou a to při instalaci aplikace.[7]



Obrázek 6: Diagram achitektury operačního systému Android (převzato z [1])

Čtvrtou vrstvou je aplikační framework. Ten poskytuje vývojářům přístup k systémovým službám používaných v aplikacích. Je tvořena sadou knihoven psaných v jazyce Java a celkově tvoří systémové API. Pomocí tohoto API může vývojář přistupovat k hardwarovým zařízením, k prvkům uživatelského rozhraní nebo poskytuje přístup k datovým zdrojům jako jsou řetězce, grafika či další soubory. Nejvyšší vrstvou architektury je pak aplikační vrstva zahrnující sa-

motné aplikace. Zahrnuje nativní aplikace jako jsou Kontakty, Mapy, Prohlížeč. Dále aplikace od vývojářů dostupné pomocí distribuční služby Google Play a aplikace třetích stran instalované ručně.

3.3 Komponenty aplikací

Android aplikace se může skládat ze čtyř základních složek, které mezi sebou s výjimkou content provideru mohou komunikovat prostřednictvím zpráv, tzv. „intentů“. Ne každá aplikace potřebuje využívat všechny tyto prvky nicméně každá z používaných komponent musí být předem definována v souboru `AndroidManifest.xml`[3], uloženém v kořenovém adresáři projektu.

3.3.1 Activity

Jedna activity, neboli „aktivita“ reprezentuje jednu obrazovku aplikace a je základním prvkem pro interakci s uživatelem. Jedna aplikace standardně obsahuje více oddělených aktivit, mezi kterými uživatel přechází při práci s aplikací. Aktivity jsou schopny předávat si data pomocí intentů. Rozložení grafických a ovládacích prvků aktivity určuje layout definovaný v XML souboru. Spuštění nové aktivity je výpočetně náročná operace a proto je při přecházení mezi aktivitami žádoucí, aby se jejich stav zachoval pro opětovné vyvolání. Tuto funkčnost obstarává Activity Manager, který je zodpovědný za životní cyklus všech aktivit v aplikaci. Má za úkol vytvářet aktivity a uchovávat informace o jejich stavu v zásobníku na jehož vrcholu je vždy aktuálně zobrazovaná aktivita.[1] Životní cyklus aktivity se může nacházet v jednom ze čtyř stavů [7]:

- launched - Ve chvíli kdy je aktivita inicializována
- running - Během zobrazení aktivity na displeji
- killed - Activity Manager aktivitu zrušil z nedostatku paměti
- shut down - Aktivita byla odstraněna ze zásobníku

3.3.2 Service

Service je komponenta běžící na pozadí aplikace a neposkytuje žádné uživatelské rozhraní. Využívá se pro vykonávání dlouhodobých nebo dlouho trvajících úkolů. Služby se využívají například pro síťovou komunikaci kde není předem známa délka odezvy serveru nebo pro přehrávání hudby. Služba může mít dvě formy, „spuštěná“ (started) a „navázaná“ (bound). Služba spuštěná jinou komponentou, většinou aktivitou, pomocí metody startService() může být ukončena buď sama sebou nebo jinou komponentou. Druhým způsobem je službu navázat na jednu nebo více komponent zavoláním metody bindService(). Služba běží do té doby dokud je navázaná alespoň jednu komponentu. [7]

[3] `http://developer.android.com/samples/BasicContactables/AndroidManifest.html`

Služby se mohou nacházet v jednom ze tří stavů:

- called - Ve chvíli kdy je služba inicializována zavoláním nebo navázáním komponenty
- running - Během vykonávání na pozadí
- shut down - Služba byla ukončena

3.3.3 Content provider

Jedná se o jednotné aplikační rozhraní umožňující přístup k datům a jejich sdílení mezi aktivitami nebo aplikacemi. Poskytovaná data mohou být uložena v souborech, SQLite databázi nebo na internetu.

3.3.4 Broadcast receiver

Broadcast reciever je komponenta bez uživatelského rozhraní sloužící k zachytávání broadcast oznámení. Na zachycená oznámení podle jejich záměru patřičně reaguje. Broadcast receiver je používán například pro zobrazení notifikací o příchozích SMS zprávách nebo nízkém stavu baterie.

3.4 Vývoj software

Společnost Google, Inc. poskytuje vývojářům balík nástrojů pojmenovaný Android SDK (Software Development Kit). Ten se skládá z následujících částí:

- Android API což jsou knihovny, poskytující vývojáři přístup k funkcím sytému.
- Dokumentace obsahuje mnoho informací o komponentách systému a příklady použití. Velká část dokumentace je dostupná online v příručce pro vývojáře.
- Vývojářské nástroje pomocí kterých je možné vytvářet, kompilovat a ladit aplikace pro operační systém Android. Patří mezi ně například Android Debug Bridge (adb), který umožňuje komunikaci mezi počítačem a Android zařízením pomocí ADB příkazů.
- Správce virtuálních zařízení a emulátor nabízí možnost vytvořit prostředí s různou konfigurací hardware, ve kterém se dá simulovat běh aplikace. Vývojář tak může zjistiti jak se aplikace chová na velkém počtu různých typů zařízení.
- Android Studio je oficiální vývojové prostředí (IDE) založené na IntelliJ IDEA. IntelliJ IDEA je speciálně vytvořena pro rozvoj Androidu a je k dispozici ke stažení pro Windows, Mac OS X a Linux.

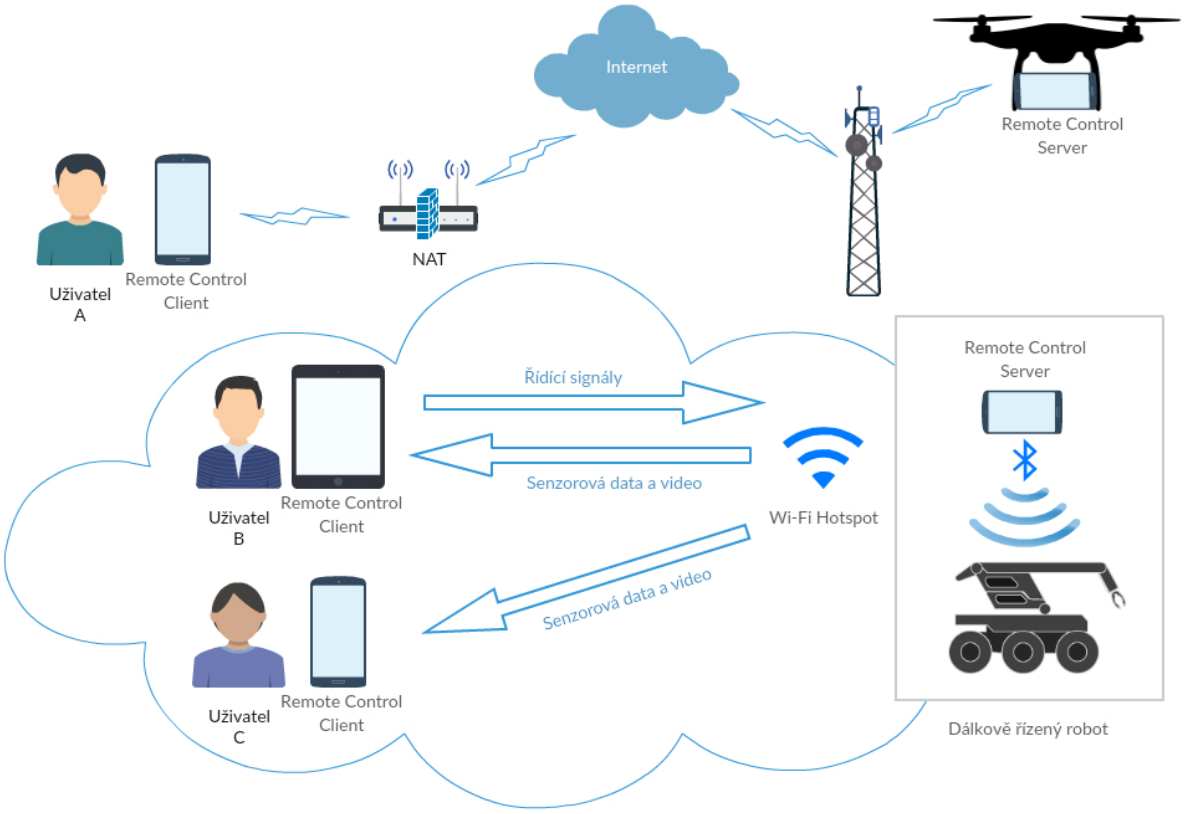
4 Řídící systém Remote Control

V rámci diplomové práce jsem vyvíjel systém `Remote Control` určený pro bezdrátové ovládání robotů a dronů pomocí mobilního zařízení. Tato kapitola podrobně popisuje vývoj systému a funkce jeho jednotlivých částí. Rozebírá problémy, se kterými jsem se při vývoji potýkal a postup jakým jsem docílil jejich vyřešení.

Řídící systém `Remote Control` se skládá ze dvou mobilních aplikací pro operační systém Android. První aplikace s názvem `Remote Control Server` tvoří serverovou část systému, která komunikuje s klientskou částí systému, aplikací `Remote Control Client`. Každá aplikace je určena pro samostatné mobilní zařízení. Princip spočívá v upevnění mobilního telefonu s aplikací Remove Control Server na tělo robota čímž se stane jeho součástí. Klientská aplikace `Remote Control Client` nainstalovaná na jiném mobilním telefonu nebo tabletu se poté může k tomuto serveru připojit a přijímat data ze senzorů mobilního telefonu na robotovi. Dále umožňuje přijímat obraz z videokamery a posílat řídící signály zpět robotovi. Komunikace mezi aplikacemi probíhá prostřednictvím internetové sítě, přičemž obě zařízení musí být v síti vzájemně dosažitelná. Toho lze docílit připojením obou zařízení do společné podsítě například vytvořením Ad-hoc Wi-Fi sítě na mobilním zařízení serverové části.

Komunikace mezi klienty a serverem probíhá přímo, bez nutnosti využít dalších síťových prvků. Mobilní telefony s OS Android většinou umožňují vytvořit vlastní přístupový bod (anglicky Access point (AP)), ke kterému se mohou ostatní uživatelé připojit a stát se tak součástí jedné lokální sítě, kde mezi sebou mohou přímo komunikovat. Znázornění tohoto typu spojení představují uživatelé `B` a `C` na obrázku 7, kteří jsou připojení k AP vytvořeném z mobilního telefonu umístěného na robotovi. V případě potřeby komunikace na větší vzdálenost se nabízí možnost využití mobilních sítí (GSM, UMTS) přičemž však vzniká problém v přímé dosažitelnosti obou zařízení v síti, jelikož se standardně mobilní zařízení připojují na Internet prostřednictvím jednoho či více síťových prvků s NAT. [4] Schéma tohoto spojení je uvedeno na obrázku 7, na kterém se uživatel `A` připojuje k Internetu přes síťový router s překladem adres (NAT) a firewallem, čímž se stává nedosažitelným pro přímou komunikaci se serverovou částí umístěnou na bezpilotním dronu. Řešení pro komunikaci skrz NAT je popsán v kapitole 4.7.

[4] Network Address Translation (NAT) je způsob úpravy síťového provozu přes router přepisem zdrojové nebo cílové IP adresy. NAT se většinou používá pro přístup více počítačů z lokální sítě na Internet pod jedinou veřejnou adresou, znemožňuje však přímou komunikaci mezi klienty a může snížit rychlost přenosu.



Obrázek 7: Grafické znázornění komunikace v systému Remote Control

4.1 Sběr dat ze senzorů

Chytré telefony jsou dnes osazeny celou řadou senzorů. Pro řídící systém jsou však klíčové především informace o poloze a pohybu, proto jsem do systému `Remote Control` zahrnul data z akcelerometru, gyroskopu a GPS.

V OS Android se k senzorům zařízení přistupuje přes systémovou službu `SensorManager`[5] jejíž instanci získáme zavoláním `Context.getSystemService()` s argumentem `SENSOR_SERVICE`. Data z konkrétního senzoru získáme zaregistrováním event listeneru (instance třídy implementující SensorEventListener) jehož metoda `onSensorChanged()` bude zavolána při každé změně stavu jakéhokoliv registrovaného senzoru. Při zachycení události o změně stavu senzoru je tedy třeba zjistit o jaký typ senzoru se jedná a podle toho s událostí naložit. Pro snadnější a jednotnější práci se senzory jsem vytvořil abstraktní třídu `AbstractSensor` obstarávající práci se službou `SensorManager` a definující metody, které musí každá třída reprezentující senzor implementovat. Přidání nového senzoru poté spočívá pouze ve vytvoření třídy rozšiřující `AbstractSensor` a implementaci metody pro identifikaci typy senzoru `getSensorType()` a zpracování dat pro daný senzor `processEvent()`. Události jsou zachycovány nepřetržitě a zpracováním události o změně je přepsán vnitřní stav objektu `AbstractSensor`. Všechny používané senzory jsou definované ve třídě `SensorController`, která se stará o vytváření jejich instancí, spouští či zastavuje snímaní dat a vrací souhrnná data ze všech spravovaných senzorů. Datový formát souhrnných dat je popsán níže v této podkapitole.

4.1.1 Akcelerometr

Akcelerometr je senzor měřící sílu zrychlení udávanou v jednotkách $m.s^{-2}$ pro každou ze tří fyzikálních os (x, y a z) znázorněnou na obrázku 8. Mezi základními funkcemi telefonu je používán například pro detekci orientace obrazovky nebo ve stabilizátoru obrazu ve fotoaparátu. Řídící systém `Remote Control` využívá data z akcelerometru pro určení sklonu a náklonu řízeného robota, k jejichž výpočtu je třeba znát hodnoty zrychlení ve všech třech osách. Hodnoty pro jednotlivé osy získáme z objektu třídy `SensorEvent` typu `TYPE_ACCELEROMETER` jako jednorozměrné pole datového typu *float*. Pořadí označení os (x, y a z) odpovídá pořadí indexů v poli *values* (0, 1 a 2). Pro určení náklonu a sklonu robota je nezbytné znát také jeho polohu vůči mobilnímu telefonu. V případě, že je mobilní telefon upevněný na robotovi ve vodorovné poloze displejem vzhůru odpovídá hodnota osy z gravitační síle. Pokud však mobilní telefon umístíme na přední část robota tak aby mohl videokamerou snímat prostor před sebou, bude gravitační síle odpovídat hodnota osy x. Hodnota gravitační síly je v Android SDK vyjádřena jako konstantní číslo 9.80665. Abych tedy tuto situaci vyřešil, zvážil jsem možnosti transformace os na straně serveru i na straně klienta. Jako vhodnější řešení jsem uvážil přepočet na straně serveru jelikož připojení klienti mají logicky zájem přijímat skutečnou polohu robota, nikoliv mobilního telefonu. Třídu `Accelerometer` jsem rozšířil o možnost nastavení jednoho ze dvou

[5] `http://developer.android.com/reference/android/hardware/SensorManager.html`

kalibračních módů. Výchozího `HORIZONTAL_MODE`, který ponechává data tak je vrací Android API a `VERTICAL_MODE` použitelný v případě, že je telefon umístěný na přední části robota s osou x ve vertikální poloze.

```
1 private float[] calibrate(float[] val) {
2   if (calibration == VERTICAL_MODE) {
3       float [] tmp = Arrays.copyOf(val, val.length);
4       val = new float[]{-tmp[2], -tmp[1], tmp[0]};
5   }
6
7   return val;
8 }
```

Výpis 2: Metoda třídy `Accelerometer` pro kalibraci polohy telefonu

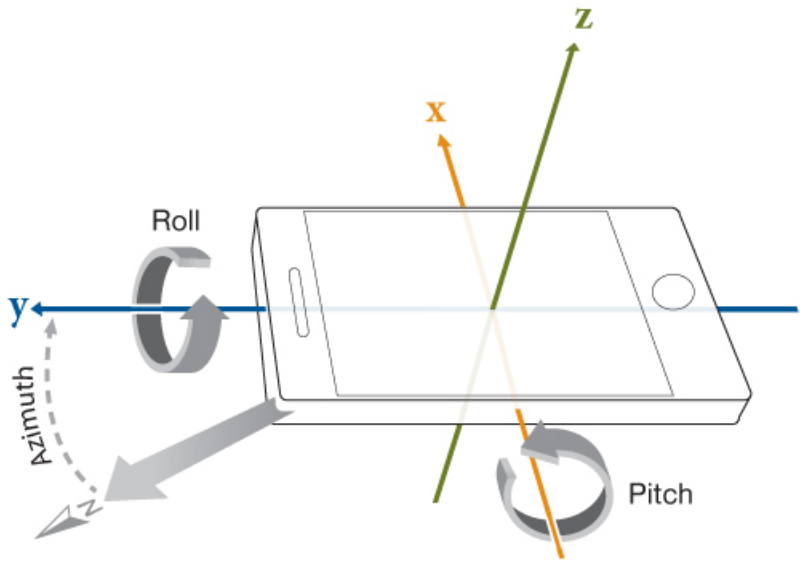
4.1.2 Gyroskop

Gyroskopický senzor určuje úhlovou rychlost mobilního zařízení ve všech jeho fyzikálních osách (x, y a z). Klasický gyroskop obsahuje setrvačník, který zachovává polohu osy své rotace v prostoru. Změnou polohy setrvačníku vzhledem ke gyroskopu je možné kalkulovat změnu polohy celého zařízení. V mobilních telefonech by použití setrvačníku bylo téměř nemožné a proto se používají mnohonásobně menší elektronické vibrační gyroskopy. Slouží tedy k podobnému účelu jako akcelerometr a je proto výhodné použít jejich kombinaci. Akcelerometr určí směr, kterým se mobilní telefon pohybuje pouze ve dvou osách. Rozpoznání pohybu i ve třetí ose zajišťuje gyroskop. Můžeme tak přesněji určit skutečný pohyb zařízení v prostoru. [35]

4.1.3 Určení polohy

V Android SDK se pro získání polohy zařízení využívá komponenty `LocationManager`[6], která poskytuje přístup ke službám pro určení polohy tzv. „location provider“. Aplikace `Remote Control Server` využívá dva hlavní „location providery“
(`NETWORK_PROVIDER` a `GPS_PROVIDER`). Vytvořil jsem třídu `GPSTracker`, která obaluje práci s `LocationManager` a snaží se získat co nejpřesnější data o poloze robota. Metoda `getLocation()` se po zavolání snaží získat data o poloze primárně pomocí služby GPS, která dokáže určit polohu zařízení s přesností do 10 metrů. Pokud se to nepodaří z důvodu chybějícího neaktivního modulu nebo nedostatečného počtu připojených satelitů, tak zkontroluje zda existují záznamy o poslední poloze metodou `getLastKnownLocation()`. Pokud je i poté poloha neznámá je stejným způsobem zavolán požadavek na `NETWORK_PROVIDER`.

[6] `http://developer.android.com/reference/android/location/LocationManager.html`



Obrázek 8: Znázornění fyzikálních os na těle mobilního telefonu (převzato z [15])

4.1.4 Senzor orientace

je softwarový typ senzoru, který měří stupeň otáčení zařízení kolem všech tří fyzikálních os. Hodnoty počítá z dat získaných z gravitačního senzoru nebo senzoru geomagnetického pole.

4.2 Aplikace Remote Control Server

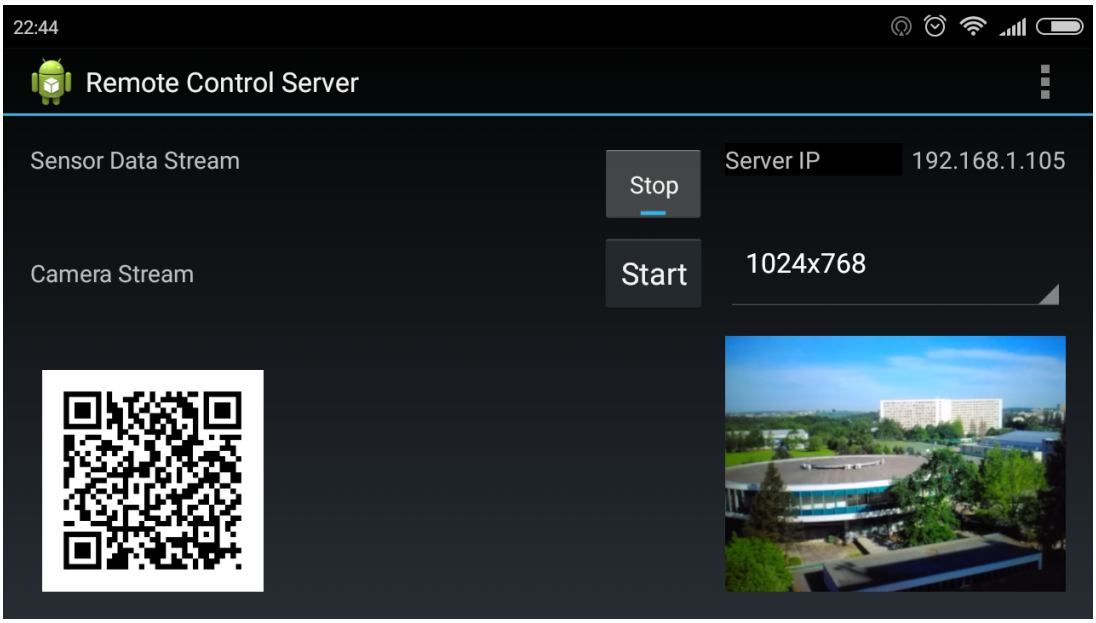
Aplikace `Remote Control Server` tvoří základní část řídícího systému `Remote Control`. Snímá data ze senzorů a videokamery mobilního telefonu upevněného na těle robota a poskytuje tato data připojeným klientům, kterých může být k serveru připojených více současně. Komunikace mobilního telefonu s tělem robota je zajištěna přes Bluetooth[7]. Při návrhu serverové části jsem bral ohled na použití starších telefonů a aplikace tedy podporována pro verzi Android 2.3.3 a vyšší.

4.2.1 Uživatelské rozhraní

Aplikace `Remote Control Server` obsahuje tři aktivity. Po spuštění aplikace je vyvolána aktivita `DeviceListActivity`, ve které má uživatel možnost vyhledat a zvolit zařízení pro vytvoření Bluetooth spojení mezi mobilním zařízením a robotem. Po výběru zařízení nebo stisknutím volby zpět dojde k přepnutí na výchozí aktivitu `MainActivity`, která obsahuje všechny hlavní prvky uživatelského rozhraní. Třetí aktivitou je `Preferences` v níž se nachází nastavení komunikace v lokální nebo veřejné síti. Na hlavní obrazovce viditelné na obrázku 9, jsou nejdůležitějšími

[7]Bluetooth je proprietární otevřený standard pro bezdrátovou komunikaci propojující dvě a více elektronických zařízení, jako například mobilní telefon, PDA, osobní počítač nebo bezdrátová sluchátka.

prvky tlačítka pro zapínání a vypínání streamu senzorových dat a videa. Tyto dvě služby je možné spouštět zvlášť jelikož běží nezávisle na sobě. Uživatel má tedy možnost například při slabém připojení zapnout pouze přenos senzorových dat, který má mnohem menší datový tok. Velikost datového toku lze ovlivnit také nastavením rozlišení streamovaného videa výběrem ze seznamu podporovaných rozlišení fotoaparátu. Dalším prvkem na hlavní obrazovce je obrázek s QR kódem (viz. 4.2.2.1) obsahujícím informace potřebné pro připojení klientské aplikace. Tato funkčnost byla popsána v kapitole 4.2.2. Po spuštění streamu videa se v pravém spodním rohu zobrazí malý náhled snímaného obrazu.



Obrázek 9: Náhled úvodní obrazovky aplikace Remote Control Server

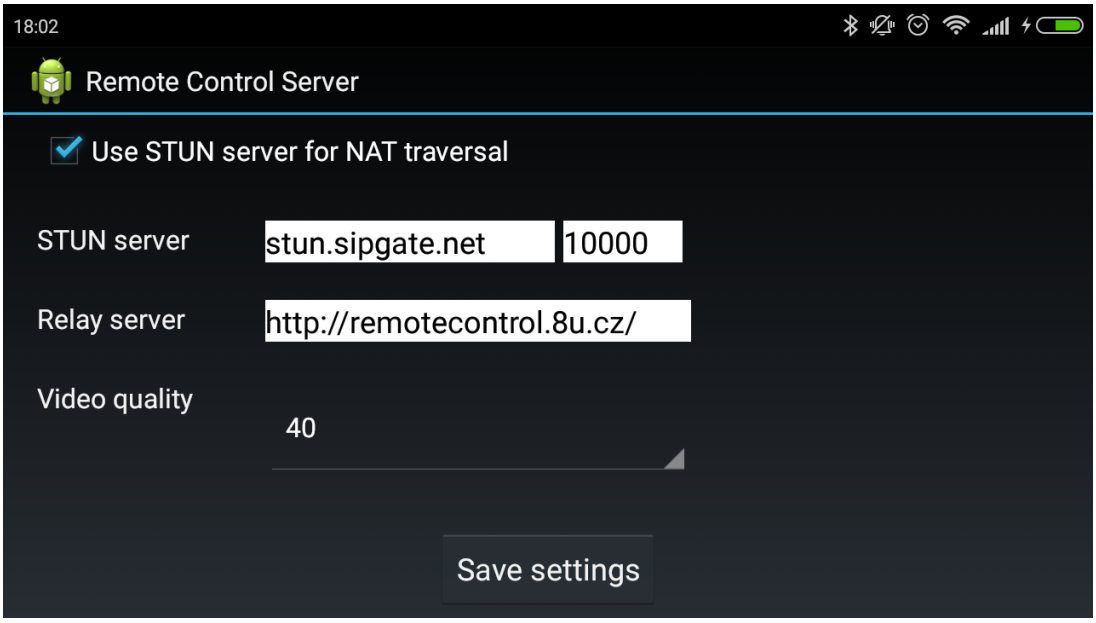
4.2.2 Nastavení komunikace

Navázání komunikace mezi aplikací `Remote Control Client` a `Remote Control Server` může probíhat dvěma způsoby. V případě, že jsou obě zařízení umístěné ve stejné síti, je možné využít přímého připojení k serveru zadáním IP adresy a čísel portů, na kterých běží naslouchání jednotlivých streamů. Tyto informace můžeme v klientské aplikaci zadat ručně ale tento způsob není příliš uživatelsky přívětivý, obzvlášť pokud je sytém využívaný různých sítích a údaje se tak často mění.

Druhým způsobem navázání komunikace je vytvoření přímého kanálu mezi serverem a klientem v případě, že jsou zařízení umístěna na jedním či víte NAT. V tom případě je potřeba předat klientovi několik údajů pro navázání spojení, které se při každém připojení mění a jejich ruční zadávání do aplikace `Remote Control Client` by bylo pracné. Elegantním řešením tohoto

problému je použití QR kódu obsahujícím veškerá data potřebná pro připojení k serveru. QR kód je generován při každé změně nastavení a zobrazován jako obrázek umístěný na hlavní obrazovce aplikace `Remote Control Server`. Aplikace `Remote Control Client` obsahuje skener QR kódu, kterým data přečte přímo z displeje serverového zařízení.

Určení způsobu navazování komunikace je možné provést v nastavení aplikace `Remote Control Server` jehož náhled je na obrázku 10. Zaškrtnutím checkboxu se zpřístupní textová pole pro zadání potřebných údajů k vytvoření komunikačního kanálu. Na internetu je veřejně dostupné velké množství STUN serverů z nichž některé uvádím v příloze 5. V nastavení je třeba zadat doménu či IP adresu serveru a číslo portu, na kterém služba běží. Do pole `Relay server` je třeba doplnit kompletní URL adresu relačního serveru včetně cesty ke skriptu. Relační server pro systém `Remote Control` byl vytvořen v rámci této diplomové práce a podrobněji je popsán v kapitole 4.7.4. Po uložení nastavení a návratu na hlavní obrazovku dojde k přegenerování QR kódu s nastavením serveru.



Obrázek 10: Náhled obrazovky s nastavením aplikace Remote Control Server

V rámci aplikací systému `Remote Control Client` jsou data s nastavením připojení a parametry serveru předávána mezi jednotlivými třídami jako instance třídy `Settings`. Předání serverového nastavení aplikaci `Remote Control Client` je tedy realizována předáním celého objektu `Settings`. Ten je na straně serveru serializován do formátu JSON a zakódován do bitové reprezentace QR kódu pomocí metody `QRCodeWriter.encode()`. Ukázka serializovaného objektu `Settings` je uvedena ve výpise 3. Ze sestrojené bitové matice je následně vytvořen obrázek zobrazitelný v `ImageView`. Generování a zpracování QR kódu není v Android API nativně pod-

porováno ale je možné využít některé z volně dostupných knihoven jako je například open-source knihovna `ZXing` [8]

```
{
   "cameraToken":"1694tcjl4l8gfpdoncdpu8g3vu",
   "stunServer":"stun.sipgate.net",
   "controlToken":"ooka5meo6kkeevdv2fo87hhvcr",
   "serverAddress":"192.168.1.106",
   "relayServer":"http://remotecontrol.8u.cz/wp-login.php",
   "sensorToken":"8bdrdo2mljcarjbi4f90lovs1q",
   "sensorUDPPort":8001,
   "controlUDPPort":8000,
   "stunPort":10000,
   "cameraUDPPort":8080,
   "useStun":true
}
```

Výpis 3: Serializový objekt třídy `Settings` do formátu JSON

4.2.2.1 QR kód je prostředek pro automatizovaný sběr dat. Stejně jako klasický EAN čárový kód jde o obrázek, který nese zakódovanou textovou informaci avšak QR kód dokáže zakódovat mnohem větší množství dat a přitom je stále velmi dobře čitelný. Kód přestává být čitelným až po odstranění či znečištění velké části kódu a není ani vyžadován vysoký kontrast barev. QR kód je dle standardu ISO 18004 tvořen bílými a černými čtvercovými body uspořádaných ve čtvercové matici, přičemž bílé body značí binární nulu a černý jedničku. Velikost výsledné matice určuje množství a typ zakódovaných dat. Definováno je celkem 40 verzí QR kódu, kde nejmenší verze (1) je tvořena čtvercovou mřížkou bodů rozměru 21x21 a největší verze (40) 177x177 bodů. Každý QR kód obsahuje trojici kotvících bodů sloužící k detekci QR kódu a určení jeho polohy vůči snímacímu zařízení ve scéně. Dále povinně obsahuje zaměřovací body a verze dva a vyšší jsou doplněny jedním nebo více vyrovnávacími body. [20]

4.3 Přenos senzorových dat

Hlavními cíli při návrhu protokolu pro přenos senzorových dat v systému `Remote Control` byla rychlost a snadná rozšiřitelnost. Vzhledem k frekvenci a objemu přenášených dat je možné zanedbat nárok na spolehlivost přenosu, kterou zajišťuje protokol TCP[9] a použít tak nespojový protokol UDP[10]. Hlavička protokolu UDP obsahuje menší množství informací než hlavička TCP

[8] ZXing (zebra crossing) je open-source knihovna pro práci s 1D a 2D čárovými kódy. Je implementována v jazyce Java a tvoří základ několika dalších projektů určených pro jiné programovací jazyky. `https://github.com/zxing/zxing`

[9] `https://tools.ietf.org/html/rfc793`

[10] `https://tools.ietf.org/html/rfc768`

a tím má tento protokol menší režii a protože UDP nemá žádné mechanismy pro dorozumívání se obou komunikujících stran je přenos rychlejší než v případě TCP. V případě ztráty nebo poškození packetu se senzorovými daty dojde na straně klienta pouze k nepatrným projevům jako například poskočení pozice robota na mapě.

4.3.1 Volba způsobu komunikace

Pro rozesílání dat v síti se nabízí více přístupů z nichž některé rozvedu v této části. Při výběru vhodného řešení jsem vycházel z cíle, umožnit zasílání senzorových dat na více klientských zařízení, které se mohou v případě připojení přes GSM síť nacházet v různých podsítích.

- **Broadcast** je jeden ze způsobů vysílání zpráv v TCP/IP síti, který používá se nespojový tzv. „connectionless“ protokol UDP. Je to vysílání jednoho všem, tedy vysílaný paket je (teoreticky) zachycen všemi zařízeními v síti, lepé řečeno v dané broadcast doméně (subnetu). Toto vysílání se používá převážně v LAN sítích (ne WAN). Tento způsob však zbytečně zatěžuje síťový provoz a pro použití v řídícím systému tak není vhodný.

- **Multicast** je způsob komunikace při níž se jedna informace doručuje skupině cílů. Multicast využívá efektivní metodu doručování, aby pakety v síti putovaly pouze jednou. Principem je, že se vytvoří mutlicastová skupina s IP adresou třídy D (224.0.0.0 – 239.255.255.255). Rozsahy IP adres třídy D rozdělujeme do tří typů uvedených v tabulce 2. Pro vytváření skupiny a registrování se používá protokol IGMP. Při směrování multicastových paketů se nepoužívají standardní směrovací protokoly, protože jednotlivé uzly patřící do multicastových skupin často vznikají a zanikají. Jedny z používaných směrovacích protokolů k identifikaci skupin náležejících k přenosu multicast a k vytváření cest pro každou skupinu jsou PIM (Protocol Independent Multicast), DVMRP (Distance Vector Multicast Routing Protocol) a MOSPF (Multicast Open Shortest Path First).[4, 12]

  Multicast je vhodné využít pro jednosměrný přenos velkého množství dat více příjemcům zároveň. V systému `Remote Control` by bylo možné realizovat tímto způsobem přenos obrazových a senzorových dat.

Název typu adresy	**Rozsah IPv4 adres**	**Popis využití**
Rezervované adresy	224.0.0/24	Pro nejmenší podsítě, hodnota TTL je pevně nastavena na hodnotu 1, proto nemůžou projít žádným routerem
Administrativní adresy	239.0.0.0/8	Pro soukromé použití v rámci jedné sítě
Veřejné adresy	224.0.1/24	Provoz může putovat napříč celým internetem, využívají například internetová rádia a televize

Tabulka 2: Typy multicastových IPv4 adres[12]

- **Unicast** je vysílání, kdy je paket zasílán jednomu cíli. Jedná se o běžnou komunikaci, kdy spolu komunikují dvě stanice. Komunikace může probíhat nespojově zasíláním UDP datagramů na konkrétní IP adresy nebo využitím spojově orientovaného protokolu TCP.

Zvolil jsem tedy unicastovou komunikaci a data jsou posílána samostatně každému klientovi, který zašle požadavek pro jejich získání. Zpracování požadavků na server zajišťuje třída `UDPServer`, která v asynchronním vlákně naslouchá na předem určeném UDP portu. Zasílání požadavků na server jsem standardizoval vytvořením třídy `Command` definující typy všech požadavků, které je server schopen vyhodnotit. Patří mezi ně požadavek pro získání senzorových dat a signály pro řízení robota. Klient tedy na své straně vytvoří instanci příkazu pro získání dat ze senzorů `new Command(Command.GET_DATA)` a serializovaný jej pošle na server kde je deserializován a vyhodnocen.

Pro serialializaci jsem využil knihovnu `Gson` vyvíjenou společností Google, která umí konvertovat Java objekty do formátu JSON. Zároveň také dokáže řetězec ve formátu JSON konvertovat do ekvivalentního Java objektu. Výhodou oproti jiným open-source knihovnám pro serializaci objektů do JSON je možnost konvertovat již existující objekty bez znalosti zdrojového kódu jejich třídy. Většina z nich navíc vyžaduje zápis Java anotací do tříd což není možné v případě, že nemáme přístup ke zdrojovému kódu třídy. Výhodou je také plná podpora Java generických typů.[21]

Po přijetí požadavku typu `Command.GET_DATA` se z objektu `SensorController` získají aktuální hodnoty všech jím spravovaných senzorů v datové zprávě a ta se pošle zpět klientovi. Datová zpráva je objekt třídy `DataMessage` obsahující hodnoty jednotlivých senzorů v kolekci polí. Třída dále konvertuje data do formátu JSON, ve kterém jsou poté posílána v UDP datagramu do klientské aplikace.

4.3.2 Návrh komunikačního protokolu

Vzhledem k tomu, že senzorová data mohou být posílána nejen v rychlých Wi-Fi sítích ale i přes pomalejší GSM, je podstatným kritériem pro přenos dat jejich objem. Rozšířeným formátem používaným ve webových službách je datový formát XML jehož použití by však vzhledem k charakteru senzorových dat (málo strukturovaná data) nepřinášelo žádné výhody. Zvolil jsem proto formát JSON, který má oproti formátu XML jednodušší strukturu zápisu a méně doplňujících dat. Zápis senzorových dat ve formátu JSON je možné vidět ve výpise 4.

4.3.2.1 JSON je datový formát, určený pro přenos dat, která mohou být organizována v polích nebo agregována v objektech. Umí pojmout indexované i neindexované pole hodnot, stavové hodnoty objektů (coby pole dvojic index:hodnota) a jednotlivé hodnoty. Výhodou datového formátu JSON je jeho nezávislost na programovacím jazyce nebo softwarové platformě. Jeho využití v systému `Remote Control` je vhodné jak pro síťovou komunikaci tak například pro přenos dat mezi jednotlivými aktivitami aplikací. Senzorová data jsou strukturována jako pole

objektů s vlastnostmi název (`name`) a hodnota (`value`), přičemž název je řetězec definovaný ve třídě `DataMessage`, který značí typ senzoru. Hodnoty senzorů jsou buď jsou zapsány jako jedna číselná hodnota nebo jako pole číselných hodnot.

```
{
   "timestamp":1459695240357,
   "sensors":[
      {
         "name":"acc"
         "value":[-9.346847, 0.153227, 0.306454],
      },
      {
         "name":"gyr"
         "value":[-0.672654, -1.413285, 0.791187],
      },
      {
         "name":"gps"
         "value":[49.46074533047323, 17.45052869506239],
      },
      {
         "name":"com"
         "value":277.0,
      }
   ]
}
```

Výpis 4: Serializovaná senzorová data ve třídě `DataMessage` do formátu JSON

4.3.3 Optimalizace přenosu senzorových dat

Jelikož komunikace mezi serverem a klientem v systému `Remote Control` probíhá nespojově přes protokol UDP je zaslání aktuálních hodnot senzorových dat klientovi vždy podmíněno příchozím požadavkem od daného klienta. Klient tedy v předem nastaveném intervalu generuje požadavky typu `Command.GET_DATA` a odesílá je na server. Při potřebě aktualizovat data v intervalu řádově stovek milisekund, stává se toto řešení neefektivním, jelikož se během krátkého časového úseku opakuje stále stejný požadavek a zatěžuje tak zbytečně síťový kanál. V tomto případě je vhodné zavést na straně serveru způsob přihlašování klientů k odběru dat. Pro tento účel jsem implementoval třídu `Multicast`, která zajišťuje správu přihlašování a odhlašování ke streamu dat. Funkce třídy `Multicast` je podrobněji popsána v kapitole 4.3.4.

Dalším ze způsobů jak snížit datový tok při streamování senzorových dat je zasílat pouze data ze senzorů, která se od posledního odeslání změnila. V případě protokolu MAVLink popsaného

v kapitole 2.6 jsou zaslány zprávy o jednotlivých změnách hodnot na senzorech zasílány zvlášť, což je při počtu různých zpráv, které MAVLink podporuje, nezbytné. Protokol systému `Remote Control` však oproti MAVLink podporuje přenos pouze malého množství rozdílných typů senzorových dat a úspora datového objemu při zasílání pouze reálně změněných senzorových hodnot by tak byla malá. V každé datové zprávě jsou tedy hodnoty ze všech senzorů.

Protokol systému `Remote Control` je možné použít i pro zasílání hodnot jednotlivých senzorů zvlášť. Zpracování zprávy v aplikaci `Remote Control Client` nevyžaduje aby objekt typu `DataMessage` obsahoval data ze všech používaných senzorů a aktualizuje stav pouze těch senzorů, pro které byla přijata data. Je tedy možné protokol využít obdobným způsobem jako funguje protokol MAVLink a zasílat data z jednotlivých senzorů ve zvláštních zprávách a to až ve chvíli kdy dojde ke změně hodnoty daného senzoru. Výhodu tohoto principu si lze snadno představit, při použití řídícího systému pro zařízení, u kterého nedochází k častým změnám senzorových hodnot, jako je například pozemní robotické vozidlo.

Nevýhodou je však použití v opačné situaci, kdy je frekvence změny senzorových hodnot vysoká a generuje se tak nadměrné množství datových zpráv, které nemusí být příjemce schopen zpracovat. Při komunikaci po kanálu s nízkou přenosovou rychlostí může dojít k jeho zahlcení. Náhlému odeslání velkého množství datových zpráv v jednom okamžiku lze předejít implementací některého z algoritmů pro řízení provozu tzv. „Traffic shaping“. Nejjednodušším algoritmem pro řízení provozu je `Leaky Bucket` jehož název i princip je odvozen od představy kbelíku s otvorem, kterým v konstantním intervalu odkapávají kapky vody. Algoritmus obdobně omezuje datový tok na pevnou přenosovou rychlost. „Kbelík“ je reprezentován frontou s omezenou délkou a pokud se zaplní, jsou další vygenerované zprávy zahozeny. Zprávy jsou z fronty postupně odebírány v určitém intervalu. [34]

4.3.4 Připojení klientů ke streamu

Připojení klientů ke streamu zajišťuje třída `Multicast`, kterou jsem vytvořil na základě principu klasického multicastového rozesílání dat. Android již od verze 1 podporuje multicast UDP datagramů díky třídě `MulticastSocket` ale pro potřeby systému `Remote Control` není tato metoda vhledem k použitým technologiím vyhovující. Problémem je především potřeba navázání spojení mezi klientem a serverem v případě kdy se nacházejí v oddělených privátních sítích za NAT či firewallem.

Principem navrženého řešení je vytvořit na straně serveru kolekci IP adres a portů klientů, kteří chtějí přijímat data ze streamu a udržovat v tomto seznamu pouze aktivní klienty, proto aby se zbytečně negenerovaly datagramy zatěžující server i síť.

Při vytváření instance třídy `Multicast` třeba do konstruktoru předat číslo portu, na kterém má běžet naslouchání příchozích požadavků o připojení k multicastu. Zavoláním metody `Multicast.open()` je v novém vlákně spuštěno naslouchání na definovaném portu a server čeká na příchozí datagram packet. Pokud zpráva sestavená z přijatého datagramu odpovídá hodnotě požadavku o připojení (hodnota je definována v konstantě `Multicast.REQUEST_JOIN`), je vy-

tvořen nový objekt typu `MulticastClient` a předán metodě `MulticastClient.join()`. Třída `MulticastClient` reprezentuje jednoho připojeného klienta, přičemž každá instance obsahuje IP adresu a port předávané v konstruktoru a informaci o čase přijetí posledního požadavku o připojení. Na základě těchto parametrů je dále vytvořena instance třídy `UDPOutputStream`[11], kterou je z objektu možné získat metodou `MulticastClient.getOutputStream()`. Před tím, než je nově vytvořený klient přidán do kolekce připojených klientů, proběhne kontrola, zda již klient se stejnou IP adresou a portem v kolekci existuje. V případě, že již klient v kolekci existuje, je mu aktualizován čas posledního požadavku o připojení a do kolekce se znovu nepřidává.

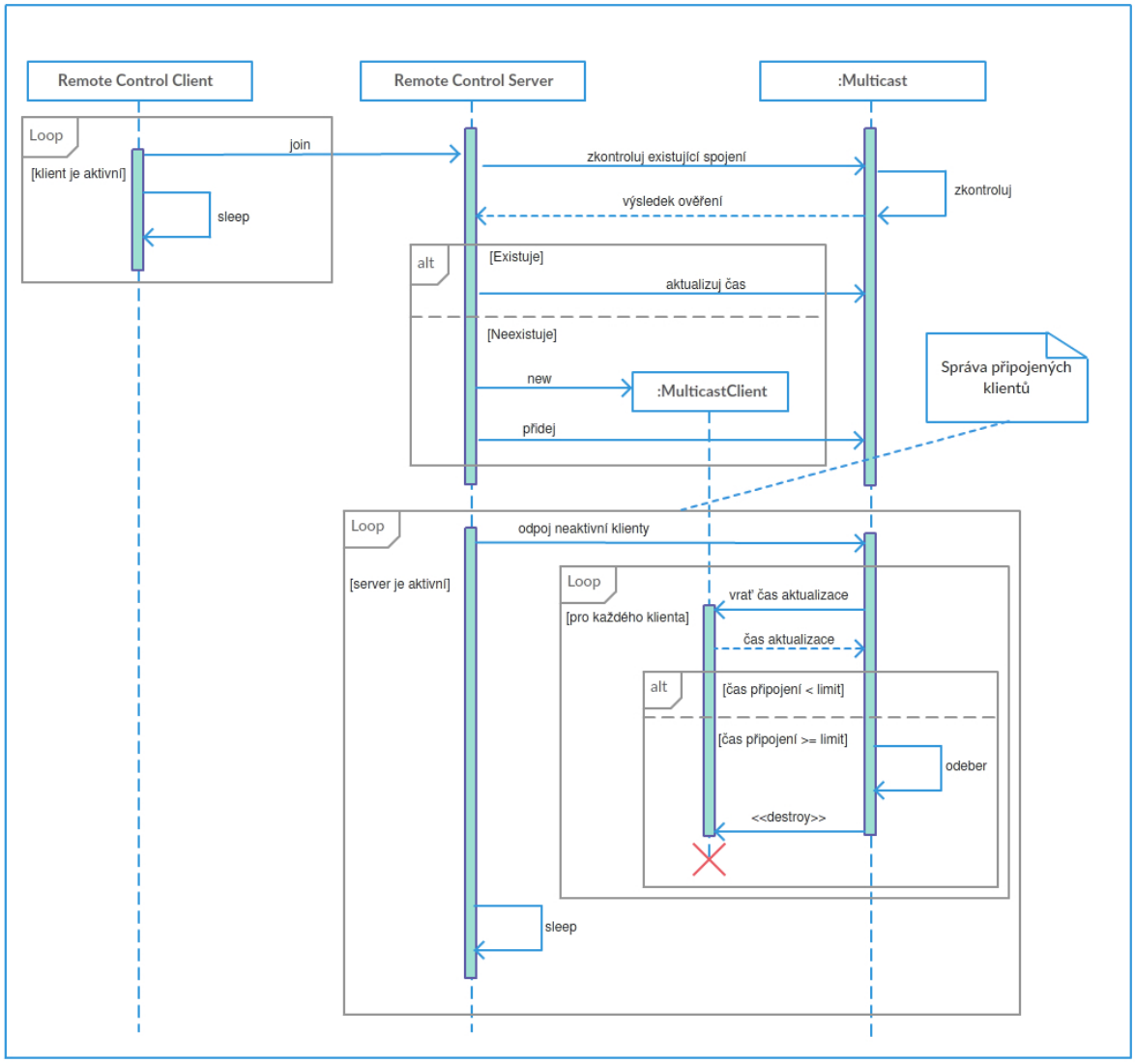
Důležitou částí třídy `Multicast` je kontrola aktivity klientů v kolekci. Tu obstarává metoda `connectionCleaner` spuštěná v separátním vlákně. V definovaném intervalu (výchozí hodnota 1 sekunda) prochází všechny připojené klienty a kontroluje zda doba od jejich posledního přihlášení nepřesáhla nastavený limit (výchozí hodnota 10 sekund). Pokud není klient aktivní, je odebrán z kolekce a přestanou se mu odesílat datagramy.

`Multicast` rozšiřuje třídu `UDPOutputStream` a překrývá všechny její metody vlastní implementací. Příklad překrytí metody `write` je uveden ve výpise 5 a je z něj patrné, že zavoláním této metody bude při průchodu kolekce klientů cyklem, zavolána metoda `UDPOutputStream.write()` na každého z nich.

```
public void write(int value) throws IOException {
  synchronized(clients) {
      for (MulticastClient client : clients) {
          client.getOutputStream().write(value);
      }
  }
}
```

Výpis 5: Překrytí metody UDPOutputStream.write() ve třídě Multicast

[11] `http://www.java2s.com/Code/Java/Network-Protocol/UDPOutputStream.htm`



Obrázek 11: Sekvenční diagram správy klientů multicastu

4.4 Kódování a přenos obrazu

V této kapitole popíši různé způsoby komprimace čistých obrazových dat do formátů vhodných pro online streamování videa a jejich dekomprimaci v operačním systému Android. Komprimace obrazových dat snižuje jejich velikost a umožňuje tak (především pro obraz ve vysokém rozlišení) rychlejší přenos po síti.

Komprimace neboli „enkódování“ se používá například při dalším zpracování dat (ukládání, streamování) z vestavěné videokamery mobilního zařízení, která získává nijak nekomprimovaná obrazová data. Opačný proces, dekomprimace „dekódování“ se používá pro získání obrazových dat z komprimovaných dat video kodeku.

Při výběru vhodného kódování pro streamování obrazu v systému `Remote Control`, byla hlavním kritériem časová prodleva mezi zachycením scény před ovládaným robotem a zobrazením na displeji klientského zařízení. Není však pravidlem, že použitím silnější komprimace bude docílena menší prodleva mezi zachycením kamery a zobrazením na displeji klienta. Komprimace a následná dekomprimace obrazových dat jsou výpočetně náročné operace, které při výkonech dnešních mobilních zařízení zaberou znatelné množství času. Silnější komprimace tedy sníží objem přenášených dat a tím urychlí jejich síťový přenos ale zvýší prodlevu.

4.4.1 MediaRecorder

Třída `MediaRecorder`[12] již od první verze OS Android (úroveň API 1) nabízí podporu pro hardwarovou akceleraci enkódování videa. Umožňuje kompresi do formátů H.263[13], H.264[14], MPEG-4[15] a VP8[16]. Hlavním účelem této třídy je nahrávání videa a zvuku z interní videokamery a mikrofonu do souboru. `MediaRecorder` nepodporuje průběžné získávání enkódovaných dat a pro sreamování videa s využitím třídy `MediaRecorder` je třeba toto omezení obejít.

Jednou z možností je průběžně ukládat krátké úseky videa do souboru a následně je načítat a zpracovávat. Nevýhodou tohoto řešení jsou výpadky obrazu způsobené v okamžiku kdy je videokamera vypnutá kvůli ukládání sekvence do souboru. Nežádoucí je také vnik značného zpoždění kvůli neustálému spouštění a ukončování záznamu i zápisu a čtení krátkých videí. Tyto nedostatky je možné částečně odstranit využitím přetížené metody `MediaRecorder.setOutputFile( FileDescriptor fd)`, díky které je možné nastavit zápis obrazových dat do instance třídy `FileDescriptor`[17]. Ta je možná získat například z instancí `FileInputStream`, `FileOutputStream` nebo přímo ze socketu. Android API umožňuje vytvoření lokálního socket serveru `LocalServerSocket`[18] a socketu `LocalSocket` čehož jde využít pro ukládání obrazových dat z objektu `MediaRecorder` místo do souboru, přímo do lokálního socketu, ze kterého je možná data přímo zpracovávat. Z bezpečnostních důvodů nelze od verze OS Android 5.0 využívat lokální socket tímto způsobem ale i toto omezení lze obejít přes třídu `ParcelFileDescriptor`[19]. Hotovou implementaci tohoto řešení včetně podpory OS Android 5.0 nalezneme například v open-source knihovně `libstreaming`[20].

[12] `http://developer.android.com/reference/android/media/MediaRecorder.html`
[13] `https://tools.ietf.org/html/rfc4629`
[14] `https://tools.ietf.org/html/rfc6184`
[15] `https://tools.ietf.org/html/rfc6416`
[16] `https://tools.ietf.org/html/rfc6386`
[17] `http://developer.android.com/reference/java/io/`
[18] `http://developer.android.com/reference/android/net/`
[19] `http://developer.android.com/reference/android/os/ParcelFileDescriptor.html`
[20] `https://github.com/fyhertz/libstreaming`

4.4.2 Třída MediaCodec

Třída `MediaCodec`[21] byla přidána v Android API úrovně 16 tedy ve verzi OS Android 4.1 a umožňuje využití nízkoúrovňového kodeku pro zvuk i video. `MediaCodec` je součástí nízkoúrovňové multimediální infrastruktury OS Android, společně s pomocnými třídami jako je `MediaExtractor`[22], `Image`[23] nebo textttAudioTrack[24]. Třída je určena pro převod čistých obrazových a zvukových dat do kompresních formátů a jejich zpětnou dekomprimaci. Podporuje například formáty H.264, H.265[25] nebo MPEG-4. [22]

Základní princip převodu dat ve třídě `MediaCodec` spočívá ve inicializaci a konfiguraci objektu buď jako enkodéru nebo dekodéru konkrétního kodeku statickými metodami třídy `MediaCodec` (`createDecoderByType()`, `createEncoderByType()` nebo `createByCodecName()`). Po konfiguraci je možné objekt spustit zavoláním metody `start`. Následně se v cyklu načítají data ze vstupního bufferu ke zpracování a samotné enkódování nebo dekódovaní probíhá cyklickým voláním metod `dequeueInputBuffer`, `queueInputBuffer`, `dequeueOutputBuffer` a `releaseOutputBuffer`. Převedená data zapisuje do výstupního bufferu nebo přímo instance třídy `Surface`[26], kterou je možné přímo vykreslit na obrazovku nebo využít pro další zpracování dekódovaných dat. Vstupní a výstupní buffery jsou tvořeny objekty `ByteBuffer` a jsou jednoduše použitelné pro stremování.

4.4.3 Formát videa H.265

Celým názvem High Efficiency Video Coding je moderní standard videa s vysokým kompresním poměrem uvedený v roce 2013. Deklaruje snížení datového toku na polovinu v porovnání s jeho předchůdcem H.264 a to při zachování stejné obrazové kvality. Má plnou podporu rozlišení až 8K (tj. UHD 7680x4320p), stereoskopické 3D zobrazení i například 12-bitové video. Kodek H.265 má tři základní profily. Základní Main je náhražka za známý H.264. Main 10 umí pracovat s 10-bitovou hloubkou obrazu, a Main Still Picture je určen pro kompresi obrázků (efektivnější než JPEG)[27, 5]

4.4.4 Formát videa MJPEG

MJPEG (Motion JPEG) je kompresní formát videa, ve kterém je každý snímek zkomprimován zvlášť jako obrázek formátu JPEG. Na rozdíl od video formátů jako [23] neboli MPEG1/MPEG2 není oficiálně standardizován a u jeho různých kodeků může dojít k vzájemné nekompatibilitě.

MJPEG může být streamován přes HTTP tak, že jednotlivé snímky jsou rozděleny do samostatných HTTP odpovědí a odděleny speciální značkou. V odpovědi na GET požadavek pro

[21] `http://developer.android.com/reference/android/media/MediaCodec.html`
[22] `http://developer.android.com/reference/android/media/MediaExtractor.html`
[23] `http://developer.android.com/reference/android/media/Image.html`
[24] `http://developer.android.com/reference/android/media/AudioTrack.html`
[25] `https://tools.ietf.org/html/rfc7798`
[26] `http://developer.android.com/reference/android/view/Surface.html`

MJPEG stream server odesílá sekvence JPEG rámců přes HTTP. Speciální typ obsahu HTTP mime-type `multipart/x-mixed-replace;boundary=<boundary-name>` informuje příjemce, že několik následujících rámců HTTP odpovědi bude odděleno pomocí řetězce `<boundary-name>`.

Motion JPEG je intraframe-only kompresní systém což znamená, že ve nevyužívá žádné techniky pro mezirámcovou predikci. Moderní interframe video formáty jako je MPEG1, MPEG2 a H.264/MPEG-4 AVC mohou dosáhnout kompresního poměru 1:50 nebo více, zatímco nedostatek mezirámcové predikce snižuje účinek komprese MJPEG kompresní poměr 1:20 a méně. Vzhledem k tomu, že jsou snímky komprimovány nezávisle na sobě, je při kompresi do formátu MJPEG vyžadováno méně hardwarových prostředků než při komprimaci do formátů s mezirámcovou predikcí.[26]

4.4.5 Streamování videa

Pro streamování videa je v aplikaci `Remote Control Server` třída `CameraStream`, která jako vstupní parametry přebírá instanci `SharedPreferences`[27] s nastavením parametrů videokamery a serveru. Jako druhý parametr je předáváno objekt třídy `SurfaceHolder`[28] reprezentující view pro zobrazení náhledu videa. Součástí předávaného nastavení může být číslo portu, na kterém má být stream dostupný (výchozí port je 8080), kvalita a rozlišení obrazu získávaného z fotoaparátu nebo zapnutí přídavného světla fotoaparátu. Zavoláním metody `CameraStream.start()` se získají aktuální hodnoty z předaného sdíleného natavení a je s nimi spuštěn `CameraStreamer` a na předaném portu je spuštěno naslouchání příchozích požadavků o připojení ke streamu. Dále je v separátním vlákně metodou `CameraStreamer.startStreamingIfRunning()` inicializován fotoaparát. Zdrojový kód inicializace fotoaparátu je uvedený ve výpise 7. V OS Android je pro získání přístupu k vestavěnému fotoaparátu a jeho vlastnostem jako automatické zaostřování, nejprve třeba přidat následující oprávnění do souboru `AndroidManifest.xml`[29].

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Výpis 6: Nastavení oprávnění pro využití služeb vestavěného fotoaparátu

[27] `http://developer.android.com/reference/android/content/SharedPreferences.html`
[28] `http://developer.android.com/reference/android/view/SurfaceHolder.html`
[29] `http://developer.android.com/samples/BasicContactables/AndroidManifest.html`

Poté je možné pracovat s třídou `Camera`[30], jenž obaluje práci se službou fotoaparátu. Jelikož mobilní zařízení může být (v dnešní době většinou je) osazeno více fotoaparáty, je prvním krokem inicializace fotoaparátu vytvoření instance konkrétního fotoaparátu předáním jeho indexu. Index fotoaparátu je celé číslo od `0` do `Camera.getNumberOfCameras() - 1`. Na řádcích 3 - 5 ve výpise 7 následuje nastavení rozlišení snímku fotoaparátu. Dále jsou získány zařízením podporované rozsahy snímkovacích frekvencí (řádek 12) a následně je na řádcích 15 - 18 instanci fotoaparátu nastaven předem zvolený rozsah snímkovací frekcence (`Parameters.PREVIEW_FPS_MIN_INDEX` až `Parameters.PREVIEW_FPS_MAX_INDEX`). Hodnoty zvoleného rozlišení pro fotoaparát jsou nastaveny (řádek 23 a 24) také náhledu získávaného ze služby fotoaparátu, což jsou zdrojová obrazová data pro stream. Na základě nastaveného rozlišení se vypočítá velikost bufferu pro obrazová vynásobením šířky, výšky a barevné hloubky náhledu. Nakonec je na instanci fotoaparátu navázán callback typu `Camera.PreviewCallback`, jehož metoda `onPreviewFrame` je zavolána při každém zobrazení snímku náhledu. Po dokončení práce s fotoaparátem je třeba zařízení opět uvolnit zavoláním metody `Camera.release()`.

Surová obrazová data jsou v obrazovém formátu YUV metodě předána jako pole bytů, které je následně enkódováno do formátu JPEG. Komprimaci obrazových dat ve formátu YUV do formátu JPEG umožňuje třída `YuvImage`[31] z grafické knihovny Android API. Komprimovaný snímek je připraven a předán pro streamování klientovi. Pro streamování obrazu ve formátu MJPEG jsem vytvořil třídu `MJpegUDPStreamer` zajišťující odesílání MJPEG snímků klientům připojeným k multicastu. Třída obsahuje dva zásobníky (`mBufferA` a `mBufferB`), do kterých jsou střídavě zapisována obrazová data předaná skrze metodu `streamJpeg`. V jedné chvíli je jeden zásobník nastaven pro zápis nově přijatého snímku a ze druhého se v odděleném vlákně čtou data pro `DataOutputStream`[32]. Tímto řešením se zamezí přepisování ještě neodeslaných snímků.

```
1  final Camera camera = Camera.open(mCameraIndex);
2  final Camera.Parameters params = camera.getParameters();
3  final List<Camera.Size> supportedPreviewSizes = params.getSupportedPreviewSizes
       ();
4  final Camera.Size selectedPreviewSize = supportedPreviewSizes.get(
       mPreviewSizeIndex);
5  params.setPreviewSize(selectedPreviewSize.width, selectedPreviewSize.height);
6
7  if (mUseFlashLight)
8  {
9      params.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
10 }
11
```

[30] http://developer.android.com/reference/android/hardware/Camera.html
[31] http://developer.android.com/reference/android/graphics/YuvImage.html
[32] http://developer.android.com/reference/java/io/DataOutputStream.html

```
12 final List<int[]> supportedPreviewFpsRanges = params.
       getSupportedPreviewFpsRange();
13 if (supportedPreviewFpsRanges != null)
14 {
15     final int[] range = supportedPreviewFpsRanges.get(0);
16     params.setPreviewFpsRange(range[Camera.Parameters.PREVIEW_FPS_MIN_INDEX],
17             range[Camera.Parameters.PREVIEW_FPS_MAX_INDEX]);
18     camera.setParameters(params);
19 }
20
21 mPreviewFormat = params.getPreviewFormat();
22 final Camera.Size previewSize = params.getPreviewSize();
23 mPreviewWidth = previewSize.width;
24 mPreviewHeight = previewSize.height;
25 final int BITS_PER_BYTE = 8;
26 final int bytesPerPixel = ImageFormat.getBitsPerPixel(mPreviewFormat) /
       BITS_PER_BYTE;
27 mPreviewBufferSize = mPreviewWidth * mPreviewHeight * bytesPerPixel * 3 / 2 +
       1;
28
29 camera.addCallbackBuffer(new byte[mPreviewBufferSize]);
30 camera.setPreviewCallbackWithBuffer(mPreviewCallback);
```

Výpis 7: Inicializace vestavěného fotoaparátu pro stereamování videa

4.5 Přenos řídících signálů

Řídící signál je informace při jejímž zpracování je na straně serveru spuštěna určitá obslužná funkce. Systém `Remote Control` umožňuje přenášet směrové řídící signály (vpřed, vzad, vlevo, vpravo) z klientské aplikace na server, kde jsou zpracovány mohou být dále přeposlány například pomocí Bluetooth do ovládaného robota. Všechny dostupné příkazy jsou definovány ve třídě `Command` jako konstanty. Konstanta určuje typ příkazu, který je předáván do konstruktoru při vytváření instance třídy `Command`. Volitelným parametrem nebo metodou `Command.setValue()` je možné nastavit hodnotu příkazu, která může v případě konkrétního směru určovat intenzitu náklonu/zatočení daným směrem.

Řídící signály jsou přenášeny jako serializované objekty třídy `Command` do formátu JSON pomocí UDP. Na serveru je spuštěno naslouchání na vyhrazeném portu pro příjem řídících signálů. Vlákno je záměrně odděleno od zpracování požadavků pro připojení ke streamu videa či senzorových dat, proto aby přenos důležitých řídících signálů nebyl omezen případným výpadkem vlákna se streamem.

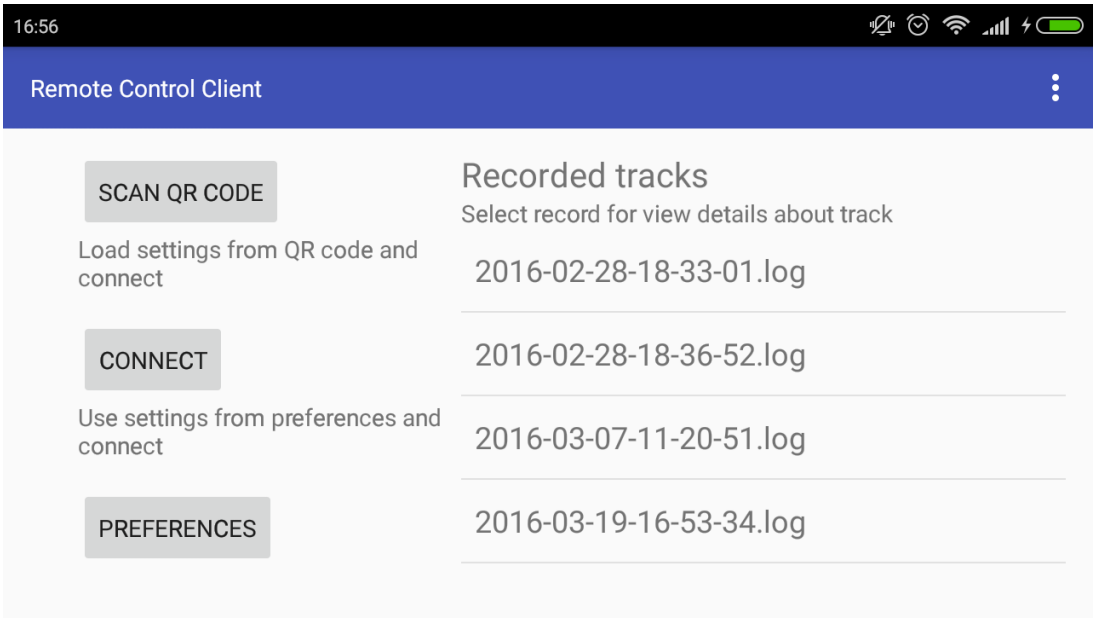
Na třídu `ControlCommandListener` je možné navázat naslouchač událostí `CommandEventListener` díky čemuž je možné přijaté příkazy zpracovávat ve vnějších třídách. Při přijetí řídícího signálu je vyvolána událost `onCommandReceived(Command command)`, kterou odchytává `MainActivity` a přes vytvořený `BluetoothSocket` předává příkazy robotickému zařízení.

4.6 Aplikace Remote Control Client

Tato kapitola popisuje aplikaci `Remote Control Client`, která tvoří klientskou část řídícího systému `Remote Control`. Účelem aplikace umožnit uživateli dálkově ovládat robota napojeného k serverové části systému a poskytnout uživateli informace aktuální o poloze robota. Vhodně tyto informace interpretovat a umožnit jejich záznam pro pozdější analýzu trasy. Aplikace vyžaduje minimální verzi Android SDK úrovně 15 co znamená, že podporuje mobilní zařízení s verzí OS Android 4.0.3 a vyšší.

4.6.1 Uživatelské rozhraní

Uživatelské rozhraní klientské aplikace je svým vzhledem a rozložením ovládacích prvků uzpůsobeno pro používání jako dálkové ovládání. Orientace obrazovky je tedy pevně nastavena do horizontální polohy. Nastavení orientace obrazovky je třeba provést pro každou aktivitu definováním atributů `android:configChanges` a `android:screenOrientation` v XML souboru `AndroidManifest.xml`.



Obrázek 12: Náhled úvodní obrazovky aplikace Remote Control Client

Úvodní obrazovka aplikace obsahuje nabídku možností připojení, zobrazení detailu uložených záznamů nebo přechodu do nastavení. Pro připojení k aplikaci `Remote Control Server` je třeba znát IP adresu serveru a čísla portů, na kterých naslouchá. Tyto informace je možné pohodlně získat načtením QR kódu z obrazovky serverové aplikace, po kterém se aplikace automaticky připojí a spustí aktivitu s ovládáním. V případě, že uživatel využívá řídící systém opakovaně ve stejné síti a adresa serveru zůstává stejná, je vhodné tyto informace zadat ručně do nastavení klientské aplikace a při dalším spuštění využít volbu `CONNECT` pro připojení s hodnotami v nastavení. Pro další vývoji aplikace se nabízí implementovat možnost uložení posledních naskenovaných dat do paměti telefonu.

Hlavní aktivitou aplikace je `ControlActivity`. Tvoří rozhraní pro ovládání robota a zobrazení aktuálních údajů o jeho poloze a pohybu. Přes celé pozadí obrazovky je rozprostřen element `MjpegView` zobrazující streamované video. Ostatní ukazatele a ovládací prvky jsou rozprostřeny po okrajích obrazovky a částečně překrývají video.

4.6.2 Čtení dat z QR kódu

Po stisknutí tlačítka `SCAN QR CODE` je spuštěna aktivita `SCAN` z externí knihovny `ZXing` obsahující čtečku QR kódů. Ta za pomoci fotoaparátu mobilního zařízení detekuje QR kód v zabírané scéně a předá jej ke zpracování. Naskenovaná data jsou ve formátu JSON vrácena zpět do `MainActivity` kde je z nich pomocí knihovny `Gson` deserializován objekt `Settings` obsahující nastavení pro připojení. Princip QR kódu je popsán v kapitole 4.2.2.1.

4.6.3 Použití externích knihoven

V projektu `Remote Control` je použito několik externích knihoven zajištujících zpětnou kompatibilitu pro starší verze OS Android, pro práci s QR kódy nebo pro serializaci a deserializaci Java objektů do formátu JSON. Ve vývojovém prostředí Android Studio je možné tyto knihovny přidat do projektu pomocí nástroje `Gradle`. Připojení externí knihovny je možné definovat v konfiguračním souboru *build.gradle*.

4.6.4 Zpracování MJPEG video streamu

Pro přijímání a zpracování video streamu je třída `VideoStream`. Jako vstup vyžaduje vykreslitelný objekt `MjpegView`, ve kterém bude video zobrazováno na displej. Dále IP adresu a port stream serveru, které budou použity pro vytvoření spojení třídou `MulticastStream`. Ta rozšiřuje třídu `UDPInputStream`[33] o správu zasílání požadavků o připojení ke streamu popsané v kapitole 4.3.4. Přijatá data jsou předána instanci třídy `MjpegInputStream`, která z přijatého datagramu vyparsuje data jednoho snímku jako pole bytů. Zobrazitelná bitmapa je následně sestavena zavoláním statické metody `BitmapFactory.decodeStream`[34].

[33] `http://www.java2s.com/Code/Java/Network-Protocol/UDPInputStream.htm`

[34] `http://developer.android.com/reference/android/graphics/BitmapFactory.html#decodeStream(java.io.InputStream)`

4.6.5 Zpracování senzorových dat

Příchozí multicast stream senzorových dat je zpracován ve třídě `SensorDataStream`. Data ze streamu jsou čtena po jednotlivých zprávách ve formátu JSON, ze kterých se vytváří datové objekty `DataMessage`. S tímto objektem dále pracuje `SensorDataInterpreter` prostřednictvím metody `processData()`. `SensorDataInterpreter` je hlavním článkem pro práci se senzorovými daty. Při vytváření jeho instance jsou inicializovány objekty používaných senzorů a dále je na něj navázán naslouchač událostí `SensorDataEventListener` předaný v konstruktoru.

Každá datová zpráva typu `DataMessage` může obsahovat data z více senzorů (záleží na počtu připojených senzorů v serverové aplikaci). V metodě `processData()` se cyklem prochází pole dat získané z datové zprávy `DataMessage.getData()`. Při každé iteraci je zjištěno o data z jakého senzoru se jedná. Data jsou reprezentována jako jednorozměrné pole datového typu *String* se dvěma hodnotami. Hodnota na indexu 0 udává typ senzoru, pod indexem 1 jsou pak dostupná samotné hodnoty ze senzoru. Hodnota typu senzoru může nabývat následujících hodnot.

- `DataMessage.TYPE_ACCELEROMETER`
- `DataMessage.TYPE_GYROSCOPE`
- `DataMessage.TYPE_COMPASS`
- `DataMessage.TYPE_GPS`

Po určení typu senzoru jdou data předána příslušné instanci v metodě `setData()`. Ta je definována v rozhraní `ISensor`, které všechny senzorové třídy musejí implementovat.

4.6.6 Vizualizace senzorových dat

Při dálkovém ovládání robota či drona jsou pro uživatele pozemní řídící stanice důležité údaje nejen o jeho pozici ale také o polohu v jaké se nachází. V případě létacího drona tyto údaje například pomáhají uživateli vyrovnávat polohu tak aby se dron nezřítil. Pro řízení pozemního robota mohou informace o sklonu a náklonu zabránit převrhnutí robota. V řídící aktivitě aplikace `Remote Control Client` jsou tyto hodnoty číselně vyjádřeny v levé horní části obrazovky a vizualizovány pomocí view `AttitudeIndicator`. Komponenty jsou viditelné na obrázku 13.

Komponenta `AttitudeIndicator` představuje umělý horizont což je letecký přístroj, který zobrazuje skutečnou vodorovnou rovinu bez ohledu na dynamické síly, působící na letadlo. Malá šipka na obvodu ukazatele polohy ukazuje směr, kterým je robot natočen vzhledem k severu. Jde o třídu rozšiřující `View`[35] což je základní stavební blok vykreslitelných částí uživatelského rozhraní. Zdrojem dat pro vykreslení komponenty je objekt třídy `Attitude` předávaný metodou `setAttitude`, která po nastavení aktuálních hodnot překreslí view zavoláním zděděné metody `invalidate()`. Třída `Attitude` je nositelem informace o poloze zařízení jako je jeho aktuální

[35] `http://developer.android.com/reference/android/view/View.html`

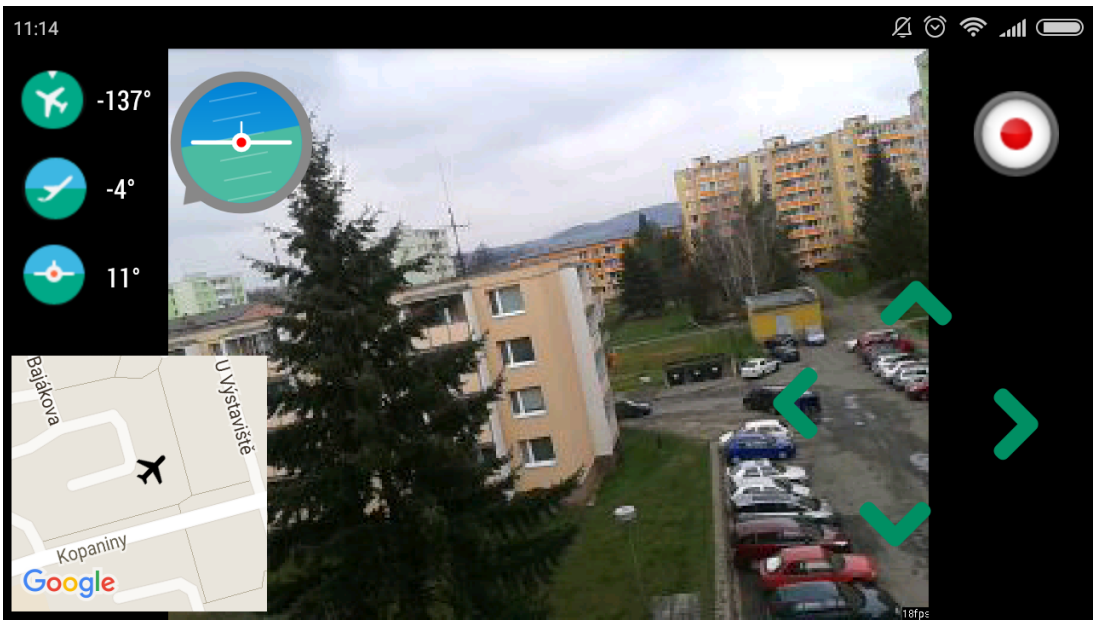
náklon (anglicky „roll“), sklon (anglicky „pitch“) a natočení (anglicky „yaw“). Tyto hodnoty se do objektu třídy předávají přímo v konstruktoru a nebo je v konstruktoru předána instance senzoru, ze které jsou získána data pro dopočítání těchto údajů.

Vzorce pro výpočet sklonu a náklonu:

$$sklon = atan(\frac{x^2}{\sqrt[2]{y^2 + z^2}})$$

$$naklon = atan(\frac{y^2}{\sqrt[2]{x^2 + z^2}})$$

Při použití těchto vzorců bude výsledek v radiánech.



Obrázek 13: Náhled úvodní obrazovky aplikace Remote Control Client

4.6.7 Zaznamenávání senzorových dat

Aplikace `Remote Control Client` umožňuje zaznamenávat tzv. „logovat“ data ze senzorů pro možnost pozdějšího přehrání trasy robota a zobrazení grafu s daty o průběhu. Při spuštěném záznamu jsou zpracovávané datové zprávy `DataMessage` předávány metodě `Logger.log()`. Třída `Logger` ukládá celé datové zprávy jako serializovaný JSON objekt společně s časem přijetí zprávy.

Android SDK nabízí tři způsoby ukládání dat. Pro ukládání relativně malých kolekcí dat jako je nastavení, je vhodné použít API `SharedPreferences`. Objekt třídy `SharedPreferences` ukazuje na soubor obsahující dvojice klíč-hodnota a nabízí jednoduché metody pro jejich čtení a zápis. Druhou možností je ukládat data do SQLite databáze s níž se v pracuje pomocí nativní

knihovny `android.database.sqlite`[36]. Použití databáze má význam pro ukládání strukturovaných dat, nad kterými potřebujeme provádět určité operace jako je například vyhledávání nebo řazení. Pro zaznamenávání senzorových dat je dostačující ukládání do souboru. Soubory mohou být ukládány do interní paměti zařízení nebo do externího úložiště (například na SD kartu).

Zaznamenávání senzorových dat v aplikaci `Remote Control Server` se nespouští automaticky ale stisknutím červeného tlačítka na ovládací obrazovce 13. Díky tomu je možné zaznamenat pouze určité části trasy robota. Vytváření souboru, zápis a ukládání obaluje třída `Logger` jejíž instance je předávána v konstruktoru třídy `SensorDataStream`. Při spuštění nového Zaznamenávání je v interním úložišti vytvořen soubor s názvem obsahujícím datum a čas vytvoření souboru (`yyyy-MM-dd-HH-mm-ss.log`) pro lepší orientaci v seznamu uložených záznamů.

4.6.8 Zobrazení uložených tras

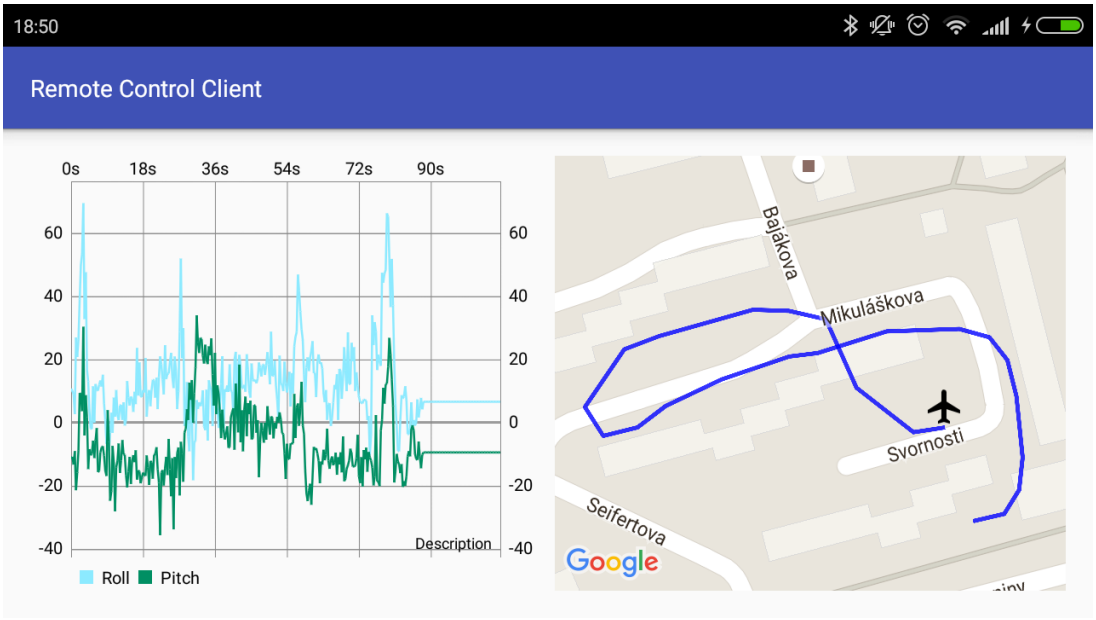
Výběrem uložené trasy ze seznamu je spuštěna aktivita `SimulationActivity`, které je ve zprávě `Intent`[37] předán název souboru zvolené trasy. Po ověření existence souboru v interním úložišti je vytvořena instance třídy `LogSource` obalující práci se souborem logu. V cyklu je poté spuštěno načítání a zpracování jednotlivých záznamů v souboru, které třída `LogSource` vrací jako entity typu `LogRecord`. Tím, že jsou příchozí data ukládána jako serializované objekty `DataMessage`, jde při načítání dat ze souboru použít stejný postup zpracování jako při streamování ze sítě. `SimulationActivity` tedy obsahuje vlastní instanci `sensorDataInterpreter`, které zpracovává datové zprávy `DataMessage` získané z jednotlivých záznamů `LogRecord`.

Pozice robota v průběhu trasy je vizualizována grafem zachycujícím v jednotlivých sekundách pohybu. Modrá křivka znázorňuje náklon robota do stran a zelená křivka jeho stoupání či klesání. Stupnice hodnot je v jednotkách stupňů. Pro vykreslení grafu jsem využil externí knihovnu `com.github.mikephil.charting.charts`[38], která nabízí několik typů grafů a nastavení.

Poloha robota na trase je zakreslena do mapy jako spojnice souřadnicových bodů. Jako mapový podklad je použito `Google Maps Android API`[39].

4.6.8.1 Google Maps Android API je knihovna, díky které je možné do Android aplikací přidávat mapy s datovým podkladem služby Google Maps. API automaticky zajišťuje přístup k serverům Google Maps, stahování dat, vykreslení mapy na obrazovku a reakci na dotyková gesta. Dále nabízí možnost přidání vlastních značek do mapy a zakreslování trasy čehož se dá v řídícím systému využít pro zobrazení uložené trasy robota.[28] Pro použití API v projektu vývojového prostředí Android Studio je třeba přidat závislost `com.google.android.gms:play-services-maps` do konfiguračního souboru `build.gradle`. Pro povolení přístupu ke službám Google Maps je třeba v souboru `AndroidManifest.xml` doplnit do meta dat aplikace přidělený API klíč. Ten je možné získat přes webové stránky společnosti Google, určené pro vývojáře. [29]

[36] `http://developer.android.com/reference/android/database/sqlite/package-summary.html`
[37] `http://developer.android.com/reference/android/content/Intent.html`
[38] `https://github.com/PhilJay/MPAndroidChart`
[39] `https://developers.google.com/maps/documentation/android-api/`



Obrázek 14: Náhled obrazovky se zobrazením uložené trasy robota

4.6.9 Simulace trasy

Díky zaznamenávání času přijetí datové zprávy je možné zpětně nasimulovat zpracování senzorových dat ve stejném pořadí a se stejnými časovými intervaly v jakých byla přijata. Pro tento účel byla vytvořena třída `Simulator`, která jako vstupní parametry vyžaduje objekt `LogSource` se zdrojovými daty a instanci `sensorDataInterpreter` pro zpracování streamovaných dat. Po spuštění metodou `Simulator.run()` je vytvořeno nové vlákno, ve kterém se postupně čtou datové zprávy a předávají se interpreteru ke zpracování. Mezi jednotlivými cykly čtení se běh vlákna pozastaví na dobu určenou rozdílem času přijetí aktuální a následující zprávy.

4.7 Komunikace skrz NAT

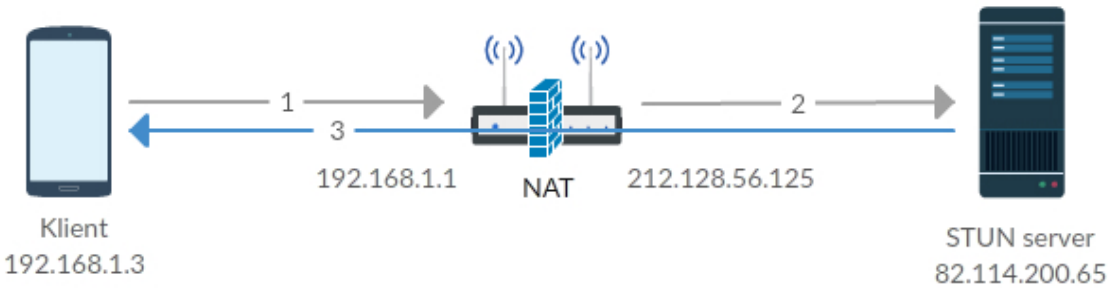
Problém v komunikaci plynoucí z umístění klientské a serverové části v oddělených podsítích byl částečně popsán v úvodu kapitoly 4. NAT je způsob překladu zdrojové nebo cílové IP adresy dle dokumentu [24] používaný v routerech, pro přístup více počítačů z lokální sítě na Internet pod jedinou veřejnou adresou. Vznikl jako důsledek omezeného počtu veřejných IP adres (IPv4 má 32 bitů a část z nich je navíc rezervována pro speciální účely). Router má přidělenu soukromou adresu, ale také je spojen s Internetem, buď jedinou veřejnou adresou, nebo několika veřejnými adresami, přidělenými poskytovatelem připojení k Internetu. Jakmile jde paket z lokální sítě do Internetu, je jeho zdrojová adresa (soukromá) přeložena na veřejnou. Router si uchovává základní data o každém aktivním spojení (adresu cíle, port). Když se vrátí odpověď na router, využije data získaná při odchozí fázi a určí kam na vnitřní síť je třeba odpověď zaslat. Pro systém na Internetu se jeví router jako zdroj i cíl komunikace. [4] Tím je však znemožněno přímo

komunikovat mezi dvěma zařízeními umístěnými za NAT bez využití společného prostředníka (serveru ve veřejně dostupném internetu).

4.7.1 STUN

`Session Traversal Utilities for NAT` (STUN) je standardizovaná [30] sada metod a internetový protokol sloužící k umožnění komunikace skrz NAT. STUN slouží například jako základ služeb pro `instant messaging` nebo `VOIP`. Dokáže zjistit typ NAT, veřejnou IP adresu (NAT adresu) a číslo portu, které NAT vyčlenilo koncovému uživateli pro UDP připojení ke vzdáleným hostitelům. Protokol vyžaduje asistenci síťového serveru třetí strany (STUN server) umístěného na opačném konci NAT, obvykle ve veřejném Internetu.[31] Rozdílné typy NAT podle RFC 5389 (STUN) jsou uvedeny níže.

Na obrázku 15 je znázorněna komunikace se STUN serverem. Požadavek na STUN server je odeslán klientem s lokální IP adresou 192.168.1.3 přes gateway s NAT, který uloží zdrojovou IP adresu a číslo portu UDP datagramu do překladové tabulky a v datagramu je nahradí za vlastní 212.128.56.125. Následně datagram odešle dál směrem k IP adrese STUN serveru. Po zpracování datagramu STUN serverem je zpět klientovi zaslána informace o IP adrese a portu, na které byly zdrojová IP adresa a port datagramu přeloženy.



Obrázek 15: Komunikace se STUN serverem

- **Full Cone** - Jedná se o typ NAT kdy jsou všechny požadavky ze stejné interní IP adresy a portu mapovány na stejnou externí IP adresu a port. Díky tomu může kterýkoliv externí hostitel zaslat paket internímu hostiteli odesláním na mapovanou externí adresu.
- **Restricted Cone** - Všechny požadavky ze stejné interní IP adresy a portu jsou mapovány na stejnou externí IP adresu a port. Na rozdíl od NAT typu „Full cone“ může externí hostitel zaslat paket internímu hostiteli pouze pokud mu interní hostitel zaslal předtím paket.
- **Port Restricted Cone** - Je podobný typu „Restricted cone“ s tím, že omezení zahrnuje také čísla portů. Externí hostitel může poslat paket na konkrétní port internímu jen pokud interní hostitel předtím zaslal z tohoto portu paket externímu hostiteli.

- **Symmetric** - Všechny požadavky ze stejné interní IP adresy a portu na specifickou IP adresu a port jsou mapovány na unikátní externí zdrojovou IP adresu a port. Jestliže to samé interní hostitel pošle paket se stejnou zdrojovou adresou a portem na jiné místo určení, je použito odlišné mapování. Pouze externí hostitel, který obdrží paket může poslat UDP paket zpět internímu hostiteli. V případě symetrického NAT vidí STUN server odlišné mapování, než to odpovídající cíli, ke kterému se mají skrze klienta zasílat pakety.

4.7.2 TURN

`Traversal Using Relays around NAT` (TURN) je podobný STUN s tím rozdílem, že zajišťuje vytvoření relace mezi klienty. Je složen ze dvou částí, `TURN server` umístěný ve veřejné síti a `TURN client` zakomponovaný v aplikaci koncového klienta. Základním principem TURN je, že `TURN client` pošle požadavek na `TURN server` „mohl by jsi pro mně alokovat relační adresu na transportní vrstvě?“ a TURN server odpoví zprávou s alokovanou IP adresou a číslem portu. TURN server má logicky rezervované IP adresy pro přidělení klientům. Jakmile TURN client obdrží IP adresu přiděleného TURN serveru může pomocí něj vytvořit relaci s dalšími klienty tzv. „endpointy“.

Nalezení optimální cesty mezi endpointy může být uskutečněno například pomocí ICE protokolu [40]. [33] ICE zprávy zasílány mezi endpointy různými cestami v síti. Jakmile projdou všechny signální trasy, vybere se na základě zjištěných informací nejvhodnější trasa pro komunikaci mezi endpointy. [6]

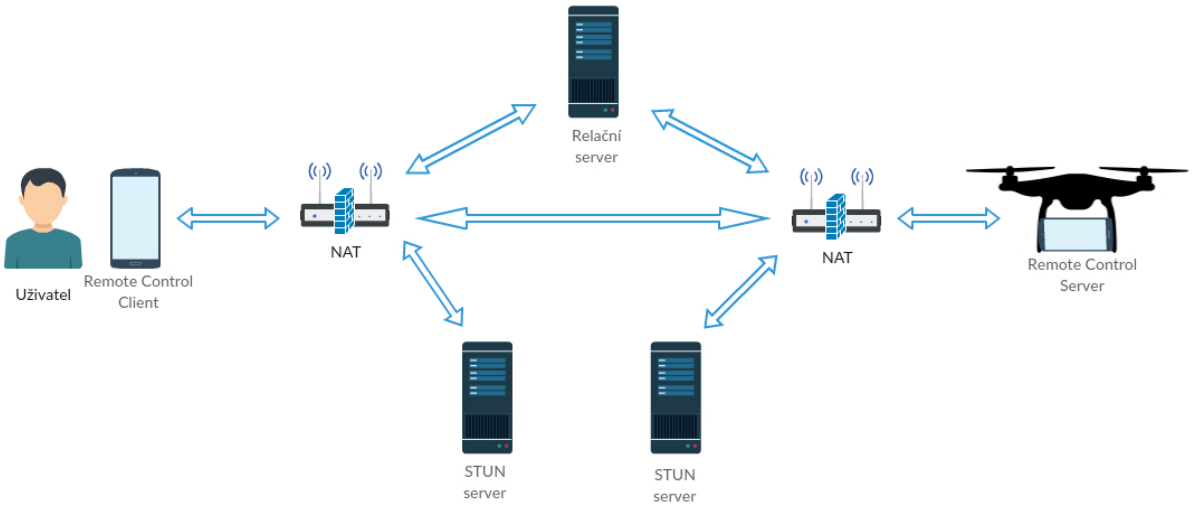
4.7.3 UDP hole punching

Vytvoření přímého UDP kanálu mezi dvěma zařízeními umístěnými za NAT se anglicky nazývá „UDP hole punching“. Tato technika většinou využívá třetí stranu přístupnou pro obě strany, která zajišťuje domluvu na síťových portech tak, aby byla umožněna přímá komunikace mezi zmíněnými dvěma konci (skrze jeden nebo více NAT). Po ustavení přímé komunikace již třetí strana není potřeba. Stávající komunikace je typicky udržována prázdnými UDP datagramy, které znemožní vypršení záznamů o dynamických překladech adres na routerech provádějících NAT, a které nemají významný vliv na zátěž datového připojení.[31, 32] Pro implementaci „UDP hole punchingu“ v aplikaci pro Android OS lze využít některého z již hotových řešení pro získání informací ze STUN serveru. Hotové implementace pro komunikaci se STUN serverem v jazyce Java jsou například projekt Stun[41] nebo JSTUN[42]

[40] `Interactive Connectivity Establishment` (ICE) je signální protokol pro usnadnění navázání komunikace mezi hostiteli umístěnými za NAT

[41] Stun - Implementace protokolu STUN podle dokumentu RFC 3489 v jazyce Java, [online] `https://java.net/projects/stun`

[42] JSTUN (Java STUN) je implementace protokolu STUN popsaného v dokumentu RFC 3489 pro jazyk Java, [online] `http://jstun.javawi.de/`



Obrázek 16: Diagram navázání přímé komunikace mezi zařízeními za NAT

Vytvoření přímého kanálu mezi aplikací `Remote Control Client` (dále v textu jako RC Client) a `Remote Control Server` (dále v textu jako RC Server) je znázorněno na obrázku 16. V první fázi RC Server vytvoří UDP spojení se STUN serverem, který zjistí IP adresu a číslo portu pro zasílání UDP datagramů aplikaci RC Server z vnější sítě. Získané údaje zašle spolu s vygenerovaným unikátním řetězcem (token) relačnímu serveru umístěnému ve veřejné síti. Relační server zajišťuje vzájemné předání informací o veřejné IP adrese a čísle portu mezi oběma stranami. Přijatá informace z RC Server je uložena pod klíčem složeným z token řetězce. Token je následně v aplikaci RC Server přidán do QR kódu mezi ostatní nastavení serveru. Po přečtení QR kódu aplikaci RC Client následuje stejný postup pro získání veřejné IP adresy a čísla portu od STUN serveru a zaslání údajů relačnímu serveru spolu se získaným tokenem z RC Server. Na základě společného tokenu relační server zjistí, kteří dva klienti spolu chtějí navázat přímé spojení a odešle jim informace o protějšku. Po obdržení zprávy z relačního serveru je třeba na port protějšího klienta zaslat UDP datagram pro vytvoření záznamu ve všech NAT na trase spojení čímž je vytvoření přímého kanálu dokončeno.

Pro vytvoření přímého UDP kanálu mezi zařízeními umístěnými za NAT slouží v sytému `Remote Control` třída `StunConnection`. Při vytváření instance je třeba konstruktoru předat adresu STUN serveru, číslo portu na kterém je služba dostupná. Dalšími parametry jsou adresa relačního serveru a řetězec identifikující relaci (token). Zavoláním metody `connect()` je spuštěn cyklus pokusů pro získání veřejné IP adresy a portu ze STUN serveru v metodě `loadMappedAddress()`. Následně jsou v metodě `createRelation()` odeslány údaje relačnímu serveru a celé vlákno čeká na odpověď s údaji o protějším klientovi. Po získání údajů je dle výše popsaného postupu vytvořen přímý kanál s protějším klientem, který je dále udržován zasíláním prázdných tzv. „keep-alive“ paketů. Třída `StunConnection` na sebe umožňuje navázat naslouchače událostí tzv. „event listener“, díky kterým se vnější objekty mohou dozvědět o

úspěšném navázání spojení (`onRelationCreated()`) a o přijetí datagramu (`onDatagramReceived(DatagramPacket incoming)`).

4.7.4 Relační server

Relační server je prostředníkem při navazování přímého spojení mezi dvěma klienty umístěnými za NAT. Jeho úloha je klíčová ale přitom jednoduchá, a proto jsem pro jeho realizaci využil HTTP server a skriptovací jazyk PHP[43]. Relační server je tvořen jedním PHP skriptem obstarávající vytvoření relace a vrácení odpovědi s IP adresou a číslem portu protějšího klienta se stejným tokenem. Jedna část sytému `Remote Control` (většinou RC Server) se HTTP serveru dotazuje na informace o svém protějšku v pravidelných intervalech, dokud své údaje relačnímu serveru nezašle i druhá část. V té chvíli HTTP server v odpovědi zašle údaje o protější straně a komunikace s HTTP serverem je ukončena. Kompletní zdrojový kód skriptu pro vytvoření relace je v příloze 10. Výhodou tohoto řešení je především snadná instalace relačního serveru na jakýkoliv webový sever podporující jazyk PHP. Na internetu existuje řada poskytovatelů základního webového hostingu s podporou PHP zcela zdarma. Ten pro potřeby systému `Remote Control` zcela postačuje.

4.7.4.1 Instalace serveru spočívá v umístění skriptu do veřejně přístupné složky webového serveru s oprávněním pro spuštění PHP skriptů. V případě použití výchozího nastavení uvedeného v 10, je třeba ve stejné složce vytvořit adresář s názvem `relations` s oprávněním pro čtení a zápis PHP serverem.

```
GET /?token=2khjhkmt87q186kboebmh54cnr&ip=77.242.90.230&port=40548 HTTP/1.1
Host: remotecontrol.8u.cz
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
User-Agent: Mozilla/5.0
Accept-Language: en-us,en;q=0.5
```

Výpis 8: Hlavička HTTP požadavku na relační server

```
{"id":"e957f759c9e0678bff017f5ab51d5369","ip":"37.48.11.217","port":"56941"}
```

Výpis 9: Odpověď relačního serveru v případě úspěšného spárování

[43]PHP (PHP: Hypertext Preprocessor) je skriptovací programovací jazyk určený především pro programování dynamických internetových stránek a webových aplikací například ve formátu HTML. `http://php.net/`

4.8 Testování systému Remote Control

Pro testování funkčnosti aplikace `Remote Control Server` jsem používal mobilní telefon Samsung GT-S5830i s verzí OS Android 2.3.6, verzí kernelu 2.6.35.7 a číslem sestavení GINGERBREAD.XXXLD3. Funkčnost aplikace `Remote Control Client` jsem testoval na mobilním telefonu Xiaomi Redmi 2 2GB s verzí OS Android 4.4.4, verzí kernelu 3.10.28-g7315b09 a číslem sestavení KTU84P. Pro testování obou aplikací jsem dále využíval mobilní telefon Samsung Galaxy S3 Mini s verzí OS Android 4.1. Při testování jsem se zaměřoval především na bezchybnou funkčnost obou aplikací a rychlost přenosu dat.

4.8.1 Dálkové ovládání automobilu

Funkčnost systému řídícího sytému `Remote Control` jsem testoval na reálném robotickém zařízení, kterým byl automobil poháněný třemi stejnosměrnými motorky. Dva zajišťují pohon zadních kol vozidla a třetí motorek zatáčí volantem. Činnost motorků je řízena jednodeskovým počítačem Arduino osazený Bluetooth modulem umožňujícím přijímání řídící signálů pro určení směru jízdy. Řídící signály jsou definovány jako znaky datového typu `char` a svými hodnotami reprezentují směry vpřed, vzad, vlevo a vpravo dle následující tabulky.

Směr pohybu	**Hodnota řídícího signálu**
vpřed	w
stop/vzad	s
vlevo	a
vpravo	d

Tabulka 3: Hodnoty řídících signálů pro určení směru pohybu vozidla

Řídící signály zaslané z řídící aplikace `Remote Control Client` jsou na straně serveru zpracovány třídou `ControlCommandListener`. Díky možnosti navázat `CommandEventListener` je možné přímo z `MainActivity` zpracovávat přijaté příkazy a následně přes vytvořenou instanci `BluetoothSocket` je zasílat do automobilu. Ten na signály ihned reaguje.

4.8.2 Analýza zatížení sítě

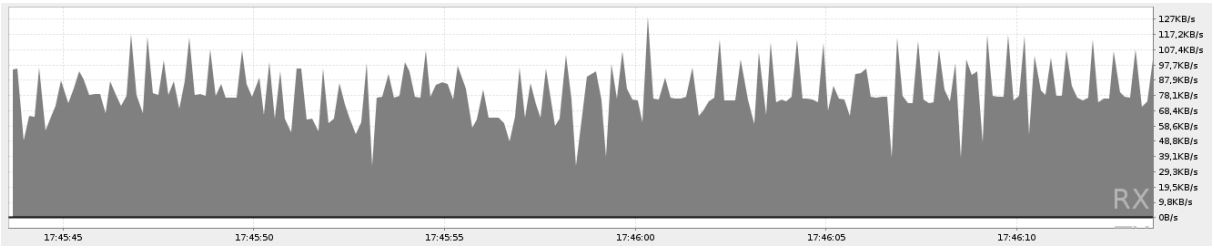
Během testování aplikace `Remote Control Server` na zařízení Samsung Galaxy S3 Mini se kvůli dobrým kvalitám interního fotoaparátu projevila potřeba nastavení kvality a rozlišení přenášeného obrazu. Jak již bylo popsáno v kapitole 4.4.4, kompresní poměr formátu MJPEG je například oproti H.265 výrazně horší ale jeho použití má význam kvůli malé prodlevě v zobrazení. Zatížení sítě je tedy značné a při použití pomalého připojení bude docházet k sekání obrazu.

Pomocí nástroje `Android Device Monitor` ze sady Android SDK jsem změřil přibližné hodnoty datového toku při různých kombinacích kvality komprimace do JPEG formátu a rozlišení přenášeného obrazu. Během všech měření zabírala videokamera stejnou scénu a FPS bylo nastaveno na 15 snímků za sekundu. Naměřené hodnoty jsou uvedeny v tabulce 4.

Rozlišení obrazu (px)	**Kvalita komprese do JPEG**						
	5%	20%	30%	40%	60%	80%	100%
144x176	25	45	65	70	85	110	300
320x240	52	68	90	112	132	195	590
480x320	62	105	146	162	188	244	700
960x720	200	341	390	420	460	700	1000

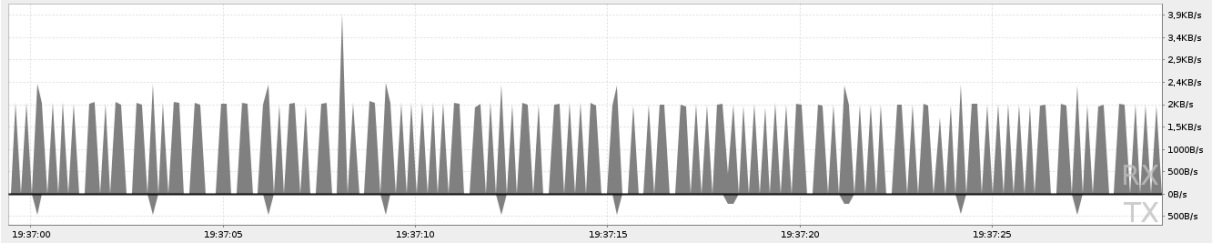
Tabulka 4: Hodnoty datového toku v KB/s přenášeného obrazu při různém rozlišení a kvalitě

`Android Device Monitor` bohužel neumožňuje podrobnější analýzu zachycených dat a průměrné hodnoty jsou tedy určeny z generovaného grafu, čímž je jejich význam pouze orientační.



Obrázek 17: Graf síťového toku při streamování videa s rozlišením 320x240 a kvalitou 40%

Naměřené hodnoty datového toku při streamování senzorových dat potvrzují tvrzení z kapitoly 4.3.3 o malé úspoře přenosu dat při zasílání pouze hodnot ze senzorů, ve kterých došlo ke změně. Na obrázku 18 je zobrazen graf zachycující datový tok senzorových dat při intervalu zasílání 200ms. Průměrná hodnota je přibližně 2KB/s což ve srovnání s nejmenším testovaných datovým tokem videa tvoří méně než 1/12 jeho objemu.



Obrázek 18: Graf síťového toku při streamování senzorových dat s intervalem 200ms

5 Závěr

Cílem této diplomové práce bylo vyvinout řídící systém složený ze dvou aplikací pro operační systém Android schopné mezi sebou přenášet řídící, senzorová a obrazová data prostřednictvím bezdrátové sítě. Vnikla tak mobilní aplikace s názvem `Remote Control Server` tvořící serverovou část systému uzpůsobenou pro instalaci do mobilního zařízení připevněného na těle robota, kterému jsou předávány řídící signály bezdrátovou technologií Bluetooth. Dále byla vytvořena aplikace `Remote Control Client` sloužící jako pozemní řídící stanice interpretující přijatá senzorová a obrazová data uživateli.

Mezi hlavní požadavky na systém patřilo minimální zpoždění přenosu dat a schopnost komunikace mezi zařízeními propojenými v rámci stejné lokální sítě Wi-Fi ale také v oddělených podsítích při použití mobilního připojení GSM/UMTS. Systém splňuje všechny stanovené požadavky a cíle, ale i přes to by pro reálné použití bylo vhodné přidat další možnosti nastavení a vyladit uživatelské rozhraní. Nabízí se například možnost přepínání mezi režimy pro pozemní vozidla a létající drony, přičemž každý režim by na kontrolní obrazovce zobrazoval ovládací prvky a hodnoty senzorů důležité pro daný typ vozidla. Aktuálně systém `Remote Control` podporuje přenos základních senzorů jejichž nabídku lze snadno rozšířit. Vylepšení by dále uvítalo vytváření komunikačního kanálu pomocí relačního serveru, které aktuálně není schopné opětovně navázat spojení v případě výpadku sítě nebo selhání vytvořeného kanálu. Zpoždění a datovou náročnost přenosu videa by bylo možné snížit automatickou volbou rozlišení a kvality obrazu založenou na sledování propustnosti sítě.

Během práce na systému `Remote Control` jsem zdokonalil své znalosti programovacího jazyka Java a získal přehled o některých technologiích a metodách, používaných při vývoji aplikací pro OS Android, se kterými jsem se doposud nesetkal. Z osobního pohledu má pro mě tato práce velký přínos a věřím, že vyvinutý systém `Remote Control` najde uplatnění v dalších projektech.

Literatura

[1] Reto Meier, *Professional Android 4 Application Development*, Wrox, 2012, ISBN-13: 978-1118102275

[2] Sayed Hashimi, *Pro Android 2*, Apress, 2010, ISBN-13: 978-1430226598

[3] Howie Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, A Bradford Book, 2005, ISBN-13: 978-0262033275

[4] Jirovský Václav, *Vademecum správce sítě*, Praha, Grada, 2001, ISBN: 80-7169-745-1

[5] Benny Bing, *Next-Generation Video Coding and Streaming*, New Jersey, John Wiley & Sons, 2015, ISBN: 978-1-118-89130-8

[6] Boon-Chong Seet, *Mobile Peer-to-Peer Computing for Next Generation Distributed Environments: Advancing Conceptual and Algorithmic Applications*, IGI Global, 2009, ISBN-13: 9781605667157

[7] *Application Fundamentals* [online]. [cit. 2016-02-26], Dostupné z: `http://developer.android.com/guide/components/fundamentals.html`

[8] *Android (operating system)* [online]. [cit. 2016-02-06], Dostupné z: `https://en.wikipedia.org/wiki/Android_(operating_system)`

[9] *Parrot AR.Drone* [online]. [cit. 2016-02-19], Dostupné z: `https://en.wikipedia.org/wiki/Parrot_AR.Drone`

[10] *AR.FreeFlight 2.4.10* [online]. [cit. 2016-02-19], Dostupné z: `https://play.google.com/store/apps/details?id=com.parrot.freeflight&hl=en`

[11] *Parrot SA - User guide* [online]. [cit. 2016-02-19], Dostupné z: `http://ardrone2.parrot.com/support/`

[12] *IP multicast* [online]. [cit. 2016-02-18], Dostupné z: `https://cs.wikipedia.org/wiki/IP_multicast`

[13] *ROBOT BLUETOOTH CONTROL* [online]. [cit. 2016-02-24], Dostupné z: `http://codetory.cz/projekty/robot-bluetooth-control`

[14] *SensoDuino: Turn Your Android Phone into a Wireless Sensors Hub for Arduino* [online]. [cit. 2016-02-24], Dostupné z: `http://www.instructables.com/id/SensoDuino-Turn-Your-Android-Phone-into-a-Wireless/`

[15] *Android Sensor Support from MATLAB* [online]. [cit. 2016-02-25], Dostupné z: `http://www.mathworks.com/hardware-support/android-sensor.html`

[16] *Sensor Ex* [online]. [cit. 2016-02-25], Dostupné z: https://sites.google.com/site/tarsensorics/home

[17] *Google Play - Sensorstream IMU+GPS* [online]. [cit. 2016-02-26], Dostupné z: https://play.google.com/store/apps/details?id=de.lorenz_fenster.sensorstreamgps

[18] *DroneKit-Android documentation* [online]. [cit. 2016-02-26], Dostupné z: http://android.dronekit.io/

[19] *MAVLink Micro Air Vehicle Communication Protocol* [online]. [cit. 2016-02-26], Dostupné z: http://qgroundcontrol.org/mavlink/start

[20] *ISO/IEC 18004:2006* [online]. [cit. 2016-03-04], Dostupné z: http://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/qr_standard.pdf

[21] *google-gson* [online]. [cit. 2016-03-10], Dostupné z: https://github.com/google/gson

[22] *Android MediaCodec stuff* [online]. [cit. 2016-03-15], Dostupné z: http://bigflake.com/mediacodec/

[23] Hoffman, et. al. *RFC 2250, RTP Format for MPEG1/MPEG2 Video* AT&T Labs - Research, 1998 [online]. [cit. 2016-04-04], Dostupné z: https://www.ietf.org/rfc/rfc2250.txt

[24] P. Srisuresh *RFC 2663, NAT Terminology and Considerations* Lucent Technologies, 1999 [online]. [cit. 2016-04-06], Dostupné z: https://tools.ietf.org/html/rfc2663

[25] Rekhter, et al *RFC 1918, Address Allocation for Private Internets* Silicon Graphics, Inc., 1996 [online]. [cit. 2016-04-06], Dostupné z: https://tools.ietf.org/html/rfc1918

[26] *Motion JPEG* [online]. [cit. 2016-03-07], Dostupné z: https://en.wikipedia.org/wiki/Motion_JPEG

[27] *Comparison of the Coding Efficiency of Video Coding Standards – Including High Efficiency Video Coding (HEVC)* J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan; T. Wiegand, [online]. [cit. 2016-03-12], Dostupné z: http://iphome.hhi.de/wiegand/assets/pdfs/2012_12_IEEE-HEVC-Performance.pdf

[28] *Google Maps Android API* [online]. [cit. 2016-03-13], Dostupné z: https://developers.google.com/maps/documentation/android-api/

[29] *Google Developers* [online]. [cit. 2016-03-13], Dostupné z: https://developers.google.com/

[30] Rosenberg, et al. *RFC 5389, Session Traversal Utilities for NAT (STUN)* Cisco, 2008 [online]. [cit. 2016-04-12], Dostupné z: https://tools.ietf.org/html/rfc5389

[31] Srisuresh, et al. *RFC 5128, State of P2P Communication across NATs* M.I.T., 2008 [online]. [cit. 2016-04-12], Dostupné z: `https://tools.ietf.org/html/rfc5128#section-3.3`

[32] *UDP hole punching* [online]. [cit. 2016-03-25], Dostupné z: `https://cs.wikipedia.org/wiki/UDP_hole_punching`

[33] J. Rosenberg *RFC 5245, Interactive Connectivity Establishment (ICE)* 2010 [online]. [cit. 2016-04-12], Dostupné z: `https://tools.ietf.org/html/rfc5245`

[34] *Leaky bucket* [online]. [cit. 2016-03-25], Dostupné z: `https://en.wikipedia.org/wiki/Leaky_bucket`

[35] *Gyro sensors - How they work and what's ahead* [online]. [cit. 2016-03-25], Dostupné z: `http://www5.epsondevice.com/en/information/technical_info/gyro/`

[36] *STUN server list* [online]. [cit. 2016-04-15], Dostupné z: `http://olegh.ftp.sh/public-stun.txt`

A Zdrojový kód relačního serveru

```php
<?php

define("LOG_PATH", "relay.log");
define("RELAY_DIR", "relations/");

function relayLog($message) {
   $log = '[' . date('d.m.Y H:i:s') . '] ' . $message . "\n";
   file_put_contents(LOG_PATH, $log, FILE_APPEND);
}

function relay($token, $ip, $port) {
   $relayFilePath = RELAY_DIR . $token;
   $relay = [];
   $clientExists = false;
   $clientId = md5($ip.$port);

   if (file_exists($relayFilePath)) {
      $relay = json_decode((file_get_contents($relayFilePath)), true);
      foreach ($relay['clients'] as $client) {
         if ($client['id'] == $clientId) {
            $clientExists = true;
         } else {
            echo json_encode($client);
         }
      }
   }

   if (!$clientExists) {
      $relay["clients"][] = [
         "id" => $clientId,
         "ip" => $ip,
         "port" => $port,
      ];

      file_put_contents($relayFilePath, json_encode($relay));
   }
}
```

```
38
39 if (isset($_GET['token']) && isset($_GET['ip']) && isset($_GET['port'])) {
40     $token = addslashes($_GET['token']);
41     $ip = addslashes($_GET['ip']);
42     $port = intval($_GET['port']);
43
44     relay($token, $ip, $port);
45     relayLog($token . "|" . $ip . ":" . $port);
46 } else {
47     relayLog("invalid request");
48 }
```

Výpis 10: PHP skript pro vytvoření relace mezi dvěma klienty umístěnými za NAT

B Seznam volně dostupných STUN serverů

stun.rynga.com:3478	stun.kundenserver.de:3478	stun.voip.eutelia.it:3478
stun.samsungsmartcam.com:3478	stun.l.google.com:19302	stun.voiparound.com:3478
stun.schlund.de:3478	stun.linea7.net:3478	stun.voipblast.com:3478
stun.services.mozilla.com:3478	stun.linphone.org:3478	stun.voipbuster.com:3478
stun.sigmavoip.com:3478	stun.liveo.fr:3478	stun.voipbusterpro.com:3478
stun.sip.us:3478	stun.lowratevoip.com:3478	stun.voipcheap.co.uk:3478
stun.sipdiscount.com:3478	stun.lugosoft.com:3478	stun.voipcheap.com:3478
stun.sipgate.net:10000	stun.lundimatin.fr:3478	stun.voipfibre.com:3478
stun.sipgate.net:3478	stun.magnet.ie:3478	stun.voipgain.com:3478
stun.siplogin.de:3478	stun.manle.com:3478	stun.voipgate.com:3478
stun.sipnet.net:3478	stun.mgn.ru:3478	stun.voipinfocenter.com:3478
stun.sipnet.ru:3478	stun.mit.de:3478	stun.voipplanet.nl:3478
stun.siportal.it:3478	stun.mitake.com.tw:3478	stun.voippro.com:3478
stun.sippeer.dk:3478	stun.miwifi.com:3478	stun.voipraider.com:3478
stun.siptraffic.com:3478	stun.modulus.gr:3478	stun.voipstunt.com:3478
stun.skylink.ru:3478	stun.mozcom.com:3478	stun.voipwise.com:3478
stun.sma.de:3478	stun.myvoiptraffic.com:3478	stun.voipzoom.com:3478
stun.smartvoip.com:3478	stun.mywatson.it:3478	stun.vopium.com:3478
stun.smsdiscount.com:3478	stun.nas.net:3478	stun.voxgratia.org:3478
stun.snafu.de:3478	stun.neotel.co.za:3478	stun.voxox.com:3478
stun.softjoys.com:3478	stun.netappel.com:3478	stun.voys.nl:3478
stun.solcon.nl:3478	stun.netappel.fr:3478	stun.voztele.com:3478
stun.solnet.ch:3478	stun.netgsm.com.tr:3478	stun.vyke.com:3478
stun.sonetel.com:3478	stun.nfon.net:3478	stun.webcalldirect.com:3478
stun.sonetel.net:3478	stun.noblogs.org:3478	stun.whoi.edu:3478
stun.sovtest.ru:3478	stun.noc.ams-ix.net:3478	stun.wifirst.net:3478
stun.speedy.com.ar:3478	stun.node4.co.uk:3478	stun.wwdl.net:3478
stun.spokn.com:3478	stun.nonoh.net:3478	stun.xs4all.nl:3478
stun.srce.hr:3478	stun.nottingham.ac.uk:3478	stun.xtratelecom.es:3478

Tabulka 5: Seznam volně dostupných STUN serverů (převzato z [36])

C Adresová struktura přiloženého DVD

Přiložené DVD obsahuje adresáře:

/Grafy	- Výsledky testování datového toku v nástroji Android Device Montior
/Relační server	- Zdrojový kód PHP skriptu pro vytvoření relačního serveru
/Remote Control Client	- Zdrojové kódy aplikace Remote Control Client
/Remote Control Server	- Zdrojové kódy aplikace Remote Control Server
/Text	- Text diplomové práce